Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN ANDROID QUE PERMITA AHORRAR COMBUSTIBLE EN VEHÍCULOS

Autor: Javier Rodríguez Guijarro

Tutor: Mario Muñoz Organero

Leganés, julio de 2013

Título: **DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN ANDROID QUE PERMITA AHORRAR COMBUSTIBLE EN VEHÍCULOS**

Autor:

Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Hace algunos años decidí embarcarme en esta aventura de convertirme en ingeniero, frente a las dificultades y el trabajo duro que se me presentaba por delante pesó más la ilusión por aprender y la determinación por demostrar que querer es poder.

En el punto en el que me encuentro actualmente, ya puedo tocar con la punta de los dedos el objetivo final. Desde esta perspectiva puedo mirar hacia atrás y recordar con orgullo como conseguí superar, a base de esfuerzo y sin considerar nunca el abandono como una posibilidad, aquellos momentos realmente duros que se presentaron ante mí y, por supuesto, lo mucho que he disfrutado recorriendo este camino.

Dicho lo cual, no podría haber llegado a este punto sin la gente que me rodea, me gustaría agradecer a todos y cada uno de ellos el apoyo mostrado incondicionalmente, espero no olvidarme de nadie.

Me gustaría empezar agradeciendo a mis padres y a mis hermanos la confianza en mí que siempre han demostrado, más incluso que la mía propia. Y más aún, haberme enseñado a seguir adelante sin dejar que las dificultades por muchas o muy pesadas que puedan llegar a ser te venzan y te convenzan de que rendirse es la mejor opción.

En segundo lugar, a mis compañeros y amigos, junto a los cuales me he enfrentado a las interminables prácticas y a los duros exámenes. Sobre todo a Pablo, Salvador, Sandra, Janine, Álvaro... Son inolvidables esas navidades trabajando en los laboratorios, ese día de la lotería que año tras año pasábamos haciendo prácticas.

Muy especialmente, quiero agradecerle a mi novia Cristina todo el apoyo y todas las fuerzas que me ha dado durante este año, no sólo para ayudarme a llevar a buen puerto este proyecto sino por todas las demás cosas que sin ella sin duda habrían conseguido doblegarme. ¡Gracias por estar siempre ahí!

Por último, quiero agradecer todo lo aprendido a aquellos que han decidido compartir sus conocimientos conmigo, a mi tutor Mario, a Sourygna y Julio por enseñarme lo que significa ser ingeniero.

Resumen

En el presente proyecto se ha desarrollado una aplicación destinada a ser empleada en aquellos dispositivos que utilicen el sistema operativo Android. La aplicación será una herramienta cuyo objetivo principal es hacer que el usuario pueda ahorrar combustible en su vehículo turismo.

Debido a la situación económica actual en países como España, la palabra "ahorro" ha cobrado una especial importancia. El ahorro se ha convertido en un hábito de las economías familiares. Por ese motivo, el proyecto actual trata de ayudar a los ciudadanos en esa tarea del ahorro.

Por otro lado, en España, aproximadamente el 65% de los usuarios de dispositivos móviles utilizan teléfonos inteligentes, de los cuales, el 80% de ellos utilizan el sistema operativo Android. Esto convierte a este sistema en un medio ideal para la difusión de aplicaciones, incluyendo aplicaciones con fines divulgativos como la nuestra.

El sistema desarrollado fue diseñado con objeto de informar al usuario en tiempo real sobre las claves de la conducción eficiente, publicadas por el Instituto para la Diversificación y el Ahorro de la Energía (IDAE). La plataforma ayudará al usuario a aplicar dichas claves en función de la situación real del vehículo en cada momento.

Para evaluar la situación real del vehículo en cada instante de tiempo, la aplicación hará uso de los datos provenientes del sistema GPS incluido en el dispositivo Android. Mediante dichos datos, la plataforma podrá analizar la conducción del usuario y corregir aquellos aspectos de la misma que supongan un gasto innecesario de combustible, además de sugerir al usuario nuevas técnicas que favorezcan el ahorro en ese gasto.

Gracias a nuestra aplicación el usuario podrá comprobar los beneficios de la conducción eficiente, entre los cuales se incluyen: aumento en la seguridad, menores costes de mantenimiento del vehículo, ahorro del gasto en combustible, etc.

Palabras clave: Smartphone, WiFi, GPS, Bluetooth, sistema operativo, Android, Google, Google Play, IDAE, XML, API, intento, ActionBar, layout, aplicación, dispositivo, pantalla.

Abstract

In this Project we have developed an application designed to be use on devices that use the Android operating system. The application is a tool whose main objective is to allow the user to save fuel in your car.

Due to the current economic situation in countries like Spain, the word “saving” has become particularly important. Saving has become a habit of family economies. For this reason, the current project aims to help citizens in the task of saving.

On the other hand, in Spain, about 65% of mobile users use smartphones, of which, 80% of them use the Android operating system. This makes this system an ideal medium for the dissemination of applications, including instructional purposes as ours.

The developed system will inform the user in real time over the keys to efficient driving, published by the Institute for Diversification and Saving of Energy (IDAE). The platform will help the user to apply those keys depending on the actual situation of the vehicle at all times.

To assess the real situation of the vehicle at each instant of time, the application will use data from the GPS system included in the Android device. Using these data, the platform can analyze the user’s driving and correct those aspects of it that involve a waste of fuel, and suggest the user new techniques that promote saving that expense.

Thanks to our application the user can see the benefits of efficient driving, among which include: increased security, lower vehicle maintenance costs, fuel cost savings, etc.

Keywords: Smartphone, WiFi, GPS, Bluetooth, operating system, Android, Google, Google Play, IDAE, XML, API, intent, ActionBar, layout, application, device, screen.

Índice general

Índice de figuras

Índice de tablas

Índice de código fuente

Capítulo 1

Introducción y objetivos

Vamos a comenzar la memoria del presente proyecto con un capítulo de introducción a este. Durante este primer capítulo describiremos la problemática que nuestro proyecto ayudará a resolver. Para ello, analizaremos la situación actual en relación a aquellos aspectos que acontecen a nuestro proyecto.

Una vez expuesto el problema a resolver, describiremos los objetivos del presente proyecto y la manera en la que alcanzaremos dichos objetivos. Además, se comentarán las distintas fases del desarrollo y los medios empleados para llevar a cabo el proyecto.

Por último, este capítulo incluye un apartado que resume la estructura general de la presente memoria, además, se añadirá un pequeño resumen del contenido de cada uno de los capítulos que la componen.

1.1 Teléfonos inteligentes

Los teléfonos inteligentes (del inglés *Smartphone*) están construidos sobre una plataforma informática móvil, esta les proporciona una mayor capacidad de procesamiento de datos y de conectividad que a los teléfonos móviles de generaciones anteriores, dichos dispositivos han generado una revolución en la telefonía móvil debido a las múltiples posibilidades que son capaces de ofrecer al usuario.

En febrero de 2012 Google publicó un estudio llamado: "*Our Mobile Planet: Global Smartphone Users*" **[1]**, en este estudio se muestran diferentes estadísticas acerca de estos dispositivos, en él podemos encontrar, por ejemplo, datos referentes a su incursión en la sociedad y a los usos que le dan los usuarios a estos dispositivos.

En la **tabla 1.1** se muestra el porcentaje de penetración en la población de los teléfonos inteligentes en algunos países, según los datos extraídos del citado estudio **[1]** que hacen referencia a octubre de 2011.

País	Porcentaje (%)
Estados Unidos	38
Reino Unido	45
Alemania	23
España	44
Francia	38
Japón	17

Tabla 1.1. Porcentaje de penetración de teléfonos inteligentes [1]

Como podemos observar en la **tabla 1.1**, España es el segundo país con mayor penetración de estos dispositivos, sólo por detrás de Reino Unido y por delante de países como Estados Unidos y Alemania.

Los teléfonos inteligentes poseen una serie de funcionalidades extra, además de ofrecer la posibilidad de comunicarse verbalmente o mediante mensajes de texto, como lo hacían sus predecesores, algunas de las funcionalidades más comunes de estos dispositivos son: *WiFi*, *Global Positioning System* (GPS), *Bluetooth*, acelerómetros, etc.

Gracias a los distintos sistemas operativos desarrollados para dispositivos móviles podemos gestionar dichas funcionalidades mediante aplicaciones, estos sistemas operativos, entre sus múltiples cualidades, poseen una muy interesante llamada multitarea, es decir, permiten al usuario tener varias aplicaciones corriendo al mismo tiempo.

A continuación, mostramos los datos de uso acerca de las aplicaciones para móviles de los usuarios que disponen de un Smartphone en los mismos países incluidos en la **tabla 1.1**, estos datos proceden también del estudio publicado por Google citado anteriormente **[1]**.

País	Número medio de aplicaciones instaladas	Número de aplicaciones de pago	Número aplicaciones usadas en los últimos 30 días
Estados Unidos	26	6	11
Reino Unido	23	6	8
Alemania	23	9	9
España	19	6	8
Francia	29	6	10
Japón	42	6	8

Tabla 1.2. Estadísticas uso de aplicaciones en los Smartphones [1]

Gracias a la **tabla 1.2**, podemos concluir que España es el país con menor número medio de aplicaciones instaladas por cada dispositivo móvil, sin embargo, España se encuentra al mismo nivel que el resto de países de la tabla en número de aplicaciones de pago y en el número de aplicaciones usadas.

El Sistema Operativo (SO) Android es el más extendido entre los teléfonos inteligentes. Este SO está basado en una versión modificada de Linux, se trata de un sistema de código abierto (del inglés *open-source*), entre sus múltiples cualidades posee la cualidad de ser un sistema multitarea. Android permite acceder a las funcionalidades principales del dispositivo usando aplicaciones, además, una aplicación cualquiera puede ser modificada libremente.

Según un informe de la consultora "*Kantar Worldpanel ComTech*" en julio de 2012 **[2]** el 84,1% de los terminales vendidos en España utilizaban Android como SO. El segundo puesto es para "*Research In Motion*" (RIM) con un 7,2%, este SO es utilizado en las *Blackberry*. A RIM le sigue "*Iphone OS*" (iOS) con el 3,2%. Por su parte, *Windows Phone* sólo ocupa un 0,2% de la cuota de mercado.

Gran parte del éxito de Android se debe a que permite desarrollar, reemplazar y también descargar aplicaciones de forma gratuita. Actualmente, el 70% de las aplicaciones disponibles en el repositorio de aplicaciones oficial de Android, llamado

"***Google Play***", son gratuitas. Recientemente, Google ha hecho oficial que el número de aplicaciones disponibles en este repositorio supera las 700.000.

1.2 Situación económica actual

La complicada situación económica actual, en países como España, está conduciendo a la población a llevar un ritmo de vida más austero que en años anteriores, esto se traduce en un control sobre los gastos mucho más estricto.

Centrándonos en nuestro país, el motivo principal para la austeridad entre la población de España es el paro. Según el instituto nacional de estadística (INE) **[3]**, en verano de 2012 el desempleo llegó al 25,02% de la población activa, lo que se traduce en 5,778 millones de parados en España. 1 de cada 4 personas en disposición de trabajar no tienen empleo.

Como consecuencia de las cifras del paro, el gasto medio de los hogares españoles bajó un 1% en 2011 respecto al año anterior, hasta alcanzar los 29.482 € por familia, con los gastos de vivienda (alquiler, reparaciones, agua, electricidad y combustibles) como única partida que creció entre los grupos con mayor peso en el total **[3]**.

Según el Instituto Nacional de Estadística, el desembolso de las familias que más subió fue el destinado a vivienda, agua, electricidad y combustibles, en 2011 fue un 4.3% mayor que en 2010.

El constante encarecimiento del precio de la energía, más concretamente, el precio de los combustibles fósiles como la gasolina, el gasoil, etc., afecta negativamente a las economías familiares. En septiembre de 2012 el precio medio de la gasolina sin plomo era de 1,498 €/litro y el del gasoil de 1,428 €/litro. En el mismo mes de 2002 la gasolina sin plomo costaba 0,828 €/litro y el gasóleo 0,703 €/litro. Como se puede observar, el gasóleo a duplicado el precio medio y la gasolina sin plomo se acerca también a esa situación.

Por lo tanto, el consumo de combustible de los vehículos a motor supone un gasto anual importante, esto conlleva que la población tienda a tratar de ahorrar en este gasto **[4]**. Este cambio de mentalidad junto con las normas europeas sobre emisiones de estos vehículos han obligado a los fabricantes de automóviles a ofrecer vehículos cuyo consumo de carburante es mucho más bajo que en los modelos anteriores de hace diez o veinte años, utilizando para ello las tecnologías disponibles actualmente en ese terreno.

1.3 La contaminación

Al margen de las cuestiones meramente económicas, hay otro factor que ha de ser tenido en cuenta a la hora de pensar en una reducción significativa del gasto de combustible de nuestro vehículo, hablamos de la contaminación.

Las emisiones de dióxido y monóxido de carbono (CO_2 y CO), compuestos de nitrógeno y azufre, partículas de plomo y otros contaminantes, resultantes de la utilización de los combustibles fósiles, suponen una fuente de contaminación que afecta de diversas formas y de manera importante a los seres humanos en particular y al medio ambiente en general.

En el sector del transporte se utilizan mayoritariamente combustibles de origen fósil (un 99% del combustible empleado), estos producen importantes emisiones de CO_2 a la atmósfera. Los vehículos turismos representan aproximadamente la mitad de las emisiones de este gas relacionadas con el transporte en España **[5]**. Según un estudio de "*Ecologistas en Acción*", la contaminación en Madrid es debida en un 80% a las emisiones generadas por los vehículos a motor **[6]**.

El incremento paulatino de concentración de este gas es responsable del denominado "efecto invernadero", este fenómeno provoca una progresiva aceleración en la evolución normal del clima a nivel global. La consecuencia inmediata del "efecto invernadero" es una subida de la temperatura media de la tierra, esto produce a su vez un aumento en el nivel del mar debido al deshielo del agua dulce acumulada en los polos. Este fenómeno provoca cambios en la climatología de todo el planeta a largo plazo **[7]**.

A parte de este fenómeno, se ha demostrado que la contaminación afecta directamente a la salud de los individuos que la padecen de forma habitual. A los agentes contaminantes antes mencionados se asocian enfermedades que provocan dificultades respiratorias, problemas oculares, enfermedades cardiovasculares y jaquecas, entre otras. Según una evaluación de la carga de morbilidad debida a la contaminación atmosférica llevada a cabo por la "Organización Mundial de la Salud" (OMS), cada año se producen más de 2 millones de muertes prematuras atribuibles a los efectos de la contaminación atmosférica urbana y de la contaminación del aire de interiores (causada por la utilización de combustible sólidos) **[8]**.

Por otra parte, Esperanza Martínez-Conde, profesora de la Universidad Complutense de Madrid, asegura que las partículas de plomo incluidas en los combustibles para aumentar su octanaje tienen efectos importantes sobre la salud, según esta profesora los efectos más graves de dichas partículas de plomo en la salud, cito palabras textuales: "son la interrupción del proceso de síntesis de la hemoglobina (proteína responsable del transporte de oxígeno desde los órganos respiratorios hasta los

tejidos) y, como manifestación subclínica, el retraso mental e hiperactividad en los niños" **[9]**.

La contaminación no sólo afecta a los seres humanos, los contaminantes corroen materiales y atacan a todo tipo de vegetación.

Con objeto de paliar el problema de la contaminación, el objetivo fijado en el Protocolo de Kioto era reducir las emisiones de una serie de gases que provocan el "efecto invernadero" en un 8% durante el período de 2008-2012 en relación con los niveles de 1990.

Hasta 2008 los límites de emisiones de CO_2 generados por los vehículos estaban sujetos a un acuerdo voluntario entre la Unión Europea (UE) y los fabricantes de automóviles (acuerdo ACEA). El objetivo de la UE era contribuir a llegar a un promedio de emisiones de este gas de 120 gramos (g) por kilogramo (Kg) de combustible en todos los nuevos vehículos turismos para el año 2012.

Teniendo en cuenta que los fabricantes no tenían intención de cumplir este acuerdo, en 2009 la comisión europea decidió obligar a una reducción de emisiones progresiva que persigue alcanzar los 95 g/Kg de media por coche fabricado por cada fabricante (regulación 443/2009). Se han establecido algunos pasos para alcanzar este objetivo:

- El porcentaje de vehículos de cada fabricante que deberán estar por debajo de la media irá creciendo progresivamente: 65% en 2012, 75% en 2013, 80% en 2014 y 100% a partir de 2015.
- Se han establecido penalizaciones económicas importantes si la media de emisiones de la flota fabricada aumenta respecto a 2012 o supera los márgenes.
- En 2020, el objetivo es que las emisiones sean de 95 g/km. A partir de 2013 se comenzará a debatir las medidas necesarias para ello.

Gracias a estas normas de obligado cumplimiento, en la UE los propios vehículos a motor de nueva generación ya ofrecen de por sí un consumo menor que los modelos anteriores, además, la forma en que el conductor utiliza el vehículo afecta directamente al consumo de este. Según un estudio conjunto de la "*Agencia Valenciana de Energía*" (AVEN) y el "*Instituto para la Diversificación y Ahorro de la Energía*" (IDAE) se ha concluido que una conducción eficiente puede reducir el consumo de combustible entre un 10 y un 25% **[10]**.

1.4 Ahorrar combustible

[10] En España, el sector transporte es el que presenta un mayor consumo de energía, sumando un 42% de la energía consumida en el país. Este sector es, asimismo, responsable de más del 60% del petróleo consumido y de un 30% de las emisiones

totales de CO_2. Dentro del sector transporte, el vehículo turismo tiene especial relevancia al totalizar el 15% de toda la energía final consumida en España.

A lo largo de los últimos años, la enorme evolución acontecida en la tecnología de los vehículos no se ha visto acompañada de la correspondiente evolución en la forma de conducir los mismos. Esto ha provocado un gran desajuste entre ambos aspectos.

En algunos países europeos (Suiza, Alemania, Holanda y Finlandia) fueron conscientes del desajuste existente y comenzaron a desarrollar y probar una serie de nuevas técnicas de conducción que se adaptasen a estas nuevas tecnologías de los vehículos.

Una vez probadas y reunidas las técnicas de la "conducción eficiente" e implementadas en los países de origen, la UE a través de la "*Comisión Europea*", participa en la difusión de las mismas a otros países de su entorno.

En España, el IDAE, actualmente se encuentra implementando y difundiendo las técnicas de la conducción eficiente para vehículos turismos.

La "conducción eficiente" es un nuevo estilo de conducción basado en una serie de sencillas técnicas, cuya aplicación (en vehículos de inyección) conlleva:

- Ahorro de carburante en torno al 15%.
- Reducción de las emisiones de CO_2 del 1.5%.
- Reducción de la contaminación ambiental.
- Reducción de la contaminación acústica (un coche a 4000 revoluciones por minuto (rpm) hace el mismo ruido que 32 coches a 2000 rpm).
- Aumento del confort de los ocupantes del vehículo.
- Ahorro en costes de mantenimiento del vehículo (sistema de frenado, embrague, caja de cambios…).
- Aumento de la seguridad en la conducción.

Actualmente, las técnicas de la conducción eficiente son una novedad y están siendo introducidas tanto en el sistema de enseñanza para la obtención del permiso de conducción, como en cursos y actuaciones encaminadas a formar a los conductores expertos. En poco tiempo, estás técnicas estarán al alcance de todos los conductores y todos podremos disfrutar de un estilo de conducción más eficiente en el uso de la energía, seguro y acorde con el medio ambiente.

1.5 Objetivos

A razón de lo descrito en apartados anteriores, el objetivo del presente proyecto es desarrollar una plataforma basada en el SO Android. La aplicación tendrá como fin

familiarizar al usuario, en tiempo real, con las 10 claves de la conducción eficiente, mientras conduce un vehículo a motor. Como las propias claves de la conducción eficiente, la aplicación estará orientada a vehículos de inyección cuyo combustible sea la gasolina o el gasóleo.

Se considera necesario, también, que el nivel de atención requerido por la interfaz sea lo más bajo posible, de forma que consigamos distraer lo mínimo e imprescindible al usuario de la tarea principal, que en este caso sería la propia conducción del vehículo. La plataforma se diseñará de modo que el conductor pueda aplicar estas técnicas de conducción eficiente sin que ello le suponga un esfuerzo significativo y así conservar el nivel de confort durante el viaje. El objetivo final de la aplicación consiste en que de una manera sencilla se pueda reducir el consumo de carburante durante la conducción, esto se traducirá directamente en un ahorro significativo en este gasto y en una menor emisión de gases contaminantes.

Como se ha comentado anteriormente, con este proyecto se construirá una aplicación basada en el SO Android, sencilla e intuitiva para el usuario. La aplicación hará uso del sistema GPS para determinar las condiciones en las que se está realizando el viaje, de esta forma obtendrá: velocidad, distancia, tiempo, aceleración y altitud.

Para que la plataforma se ciña lo máximo posible a la realidad del vehículo, se prestará especial atención a la precisión en los datos obtenidos, no se aceptarán datos del GPS que no mantengan un error por debajo de un límite aceptable. Por esta razón, se desestima la localización a través de la red de telefonía y el WiFi ya que en ambos casos el error obtenido ronda los 100 metros, margen intolerable para una aplicación de este tipo.

Con los datos obtenidos y siguiendo las recomendaciones para una conducción eficiente publicadas por el IDAE, el sistema indicará al usuario en qué medida está realizando una conducción eficiente, además, realizará indicaciones que el usuario podrá seguir para llevar a cabo dicha conducción.

La aplicación será válida para todos los automóviles independientemente de cuál sea la marca o el modelo, aunque dicha plataforma estará optimizada para vehículos con cambio manual y 5 marchas en su caja de cambios, sea cual sea su antigüedad, además, en vehículos con un consumo medio más alto (vehículos de mayor cilindrada) el uso adecuado de la aplicación producirá un ahorro mayor.

Teniendo en cuenta las limitaciones del dispositivo sobre el que esta aplicación va a correr, es uno de los objetivos principales de la aplicación que esta trabaje con el mayor número posible de las recomendaciones del IDAE sobre conducción eficiente. Principalmente, estas limitaciones son debidas a la imposibilidad de acceder a ciertos datos del vehículo, como pueden ser: las revoluciones del motor, la marcha engranada en la caja de cambios, posición de los pedales, etc.

Ya que la aplicación estará diseñada para correr sobre dispositivos móviles alimentados por baterías, una de las premisas tenidas en cuenta durante el diseño será, precisamente, el consumo de energía realizado por la plataforma. Como se comentó en un párrafo anterior, la aplicación hace uso del GPS instalado en el terminal, es conocido que esta funcionalidad es una de las que más batería consume en un dispositivo móvil, para contrarrestar esta cualidad de la aplicación, tanto la interfaz gráfica como el código fuente de la aplicación se diseñarán para que su consumo de energía sea el más bajo posible.

De cara a que el usuario pueda atender el interfaz de nuestra plataforma sin que afecte a su seguridad, las indicaciones de la aplicación hacia el conductor deben ser claras y fáciles de comprender e interpretar. Para que el usuario, además, no tenga que apartar los ojos de la carretera mientras conduce se establecerán indicaciones sonoras a parte de las visuales. Estas indicaciones sonoras no deben suponer una molestia para el usuario y, por tanto, estará limitado el número de estas por período de tiempo.

Otro factor importante a tener en cuenta es que la aplicación debe ser adaptable a cualquier tipo de usuario y también a las distintas necesidades de un mismo usuario. El sistema permitirá establecer los límites para la conducción de forma personalizada, no obstante, tendrá definidos unos límites aconsejables basados en los estudios antes citados sobre conducción eficiente.

La aplicación tendrá un carácter divulgativo de modo que hacerla llegar al mayor número de personas es un objetivo prioritario, la plataforma, por lo tanto, debe ser compatible con el mayor número posible de dispositivos que utilicen Android. Para ello, se buscará que el nivel del API mínimo necesario para ejecutarla sea lo más bajo posible, primando la compatibilidad por encima del diseño de la interfaz gráfica.

En el mismo apartado, referente a la compatibilidad, otro factor importante es la gran variedad de pantallas que poseen los dispositivos, diferentes tanto en tamaño como en densidad (píxeles/pulgada). Se hará uso de las herramientas que pone Android a disposición del desarrollador para solventar este problema y de esta forma conseguir que la aplicación pueda correr mostrando la misma apariencia de su interfaz tanto en una pantalla pequeña como en una grande, por ejemplo, la pantalla de una tableta (del inglés *tablet*).

A la hora de desarrollar la aplicación, se seguirán todas las recomendaciones que los desarrolladores de Android exponen en su página oficial para asegurar características muy importantes como son: la rapidez, robustez, seguridad, etc.

Por último, la plataforma no influirá en las funciones principales del dispositivo, es decir, debe permitir atender una llamada telefónica durante la ejecución de esta sin que la aplicación genere interrupciones de ningún tipo, del mismo modo, debe permitir recibir mensajes, etc.

1.6 Fases del desarrollo

Para llevar a cabo este proyecto se dividió el mismo en varias tareas o fases, estas son las tareas de las que se compone el proyecto, ordenadas de forma temporal, es decir, la primera tarea es la primera que se realizó:

- **Formación inicial (documentación previa)**: En esta fase el autor del proyecto adquirió los conocimientos necesarios para poder programar una aplicación compleja en Android. Además, dentro de esta tarea se incluye la instalación de las herramientas necesarias para el desarrollo de aplicaciones y la resolución de algunos ejemplos consistentes en desarrollar aplicaciones sencillas. Durante esta fase de documentación previa está incluida la labor de investigación sobre conducción eficiente.

- **Análisis de requisitos**: Se trata de decidir qué queremos que haga la aplicación (sus funcionalidades) y como queremos que lo haga. Sobre estas decisiones tomadas inicialmente se llevó a cabo toda la implementación de la plataforma.

- **Diseño de la aplicación**: En esta fase se diseñó toda la estructura interna del sistema, los interfaces de usuario y todos los algoritmos necesarios para realizar el trabajo descrito en la tarea anterior.

- **Implementación y pruebas unitarias de la aplicación**: En esta tarea se transformaron en código fuente los diseños de la tarea anterior, estos se desarrollaron por separado de forma que se pudieran realizar pruebas independientes (pruebas unitarias) para cada elemento desarrollado antes de integrar el conjunto.

- **Pruebas de sistema**: Durante esta fase se realizaron pruebas del sistema completo con el objetivo de verificar que este cumple con todos los objetivos descritos en el apartado de requisitos.

- **Documentación**: Esta tarea consiste en realizar toda la documentación del presente proyecto, desde comentar con precisión el código fuente hasta realizar la presente memoria.

1.7 Medios empleados

En esta sección describiremos brevemente los medios, tanto hardware como software, empleados para la realización de este proyecto, incluyendo los necesarios para desarrollar la aplicación y para generar la presente memoria.

Los elementos hardware:

- Un ordenador personal (**PC**): Se utilizó un ordenador portátil *Dell Inspiron* que contiene un procesador de doble núcleo (del inglés *Dual Core*) y 64 bits. Se ha empleado tanto para el desarrollo y test de la aplicación como para la realización de esta memoria.

- Un teléfono **Smartphone**: En este caso, para realizar los test de la aplicación desarrollada se utilizará un dispositivo *Samsung Galaxy Ace*.

Los elementos software utilizados:

- *Microsoft **Windows 7** Home Edition* de 64 bits: Como sistema operativo del equipo portátil se utilizó esta plataforma para realizar la memoria del presente proyecto.

- ***Linux Ubuntu*** versión 9.10 (*Karmic Koala*) de 64 bits: Este sistema operativo fue empleado en el ordenador personal para el desarrollo de la aplicación.

- SO ***Android*** versión 2.3.3 (*Gingerbread*): En el dispositivo móvil empleado el SO utilizado es esta versión de Android.

- ***Eclipse** Java EE IDE*, versión *Indigo*: Este entorno de desarrollo fue utilizado para realizar la aplicación.

- *Android Development ToolKit* (**ADT**) para *Eclipse*, versión 18. Para desarrollar la aplicación fueron necesarias las herramientas de desarrollo de Android para *Eclipse*.

- *Android Software Development Kit* (**SDK**), revisión 19. Este software permite compilar y depurar aplicaciones Android, además de contener las librerías propias de Android.

- *Java Development Kit* (**JDK**), version 1.5. Necesitamos el kit de desarrollo de aplicaciones Java ya que este lenguaje es la base de las aplicaciones de Android, para programar en Android utilizaremos Java añadiendo las librerías de este SO.

- ***GIMP***, versión 2.6.7: para la edición de imágenes e iconos utilizados en la aplicación.

- *Microsoft* ***Office 2007***: la presente memoria se ha creado utilizando diferentes partes de este paquete de herramientas.

1.8 Estructura de la memoria

En este apartado describiremos brevemente el contenido de los capítulos posteriores incluidos en la presente memoria.

- **Capítulo 2, estado de la cuestión**: En este capítulo hablaremos un poco de Android en general y describiremos las librerías empleadas en nuestra aplicación. Por otro lado, se detallarán las claves de la conducción eficiente. Además, para concluir el capítulo, analizaremos las aplicaciones que pueden considerarse competencia directa de la nuestra.

- **Capítulo 3, análisis, diseño e implementación**: En este capítulo describiremos el trabajo realizado para desarrollar la aplicación, desde los requisitos y casos de uso hasta el diseño e implementación de la aplicación en general y de los algoritmos más importantes de esta en particular.

- **Capítulo 4, manuales**: Este capítulo incluirá los manuales de usuario y de instalación de nuestra aplicación.

- **Capítulo 5, resultados**: Será el objetivo de este capítulo demostrar el correcto funcionamiento de la aplicación desarrollada, así como, comprobar que esta cumple con los requisitos previstos.

- **Capítulo 6, conclusiones y líneas futuras**: En este capítulo incluiremos las conclusiones finales extraídas de la aplicación desarrollada y de la memoria realizada, por otro lado, se citarán algunas futuras posibles mejoras para la aplicación.

- **Capítulo 7, planificación y presupuesto**: En este capítulo se incluye la planificación inicial y final del presente proyecto, además, se detalla el presupuesto del mismo. Por último, en este capítulo encontramos una sección que incluye información sobre cómo podemos poner a la venta nuestra aplicación para recuperar la inversión inicial.

Capítulo 2

Estado De La Cuestión

Después del capítulo de introducción vamos a continuar la presente memoria con un capítulo que va a mostrar el estado del arte. A lo largo de este capítulo describiremos las tecnologías empleadas para la realización de la aplicación, además, se comentarán las claves para realizar una conducción eficiente (energéticamente hablando) que aplicará el sistema desarrollado. Al final del capítulo encontraremos una pequeña lista de aplicaciones existentes que pertenecen al mismo ámbito que la nuestra y que pueden, por lo tanto, considerarse competencia directa de nuestra aplicación.

2.1 Android

En este apartado hablaremos del SO Android, comenzaremos hablando de este SO a nivel general y poco a poco nos iremos introduciendo en la estructura y los componentes de una aplicación Android. Al final de este apartado veremos un poco más en detalle aquellas partes de Android que se emplearán en nuestra aplicación.

2.1.1 ¿Qué es Android?

[11] Android es un SO orientado a dispositivos móviles, basado en una versión modificada del núcleo Linux (del inglés *Linux Core*). Inicialmente fue desarrollado por "*Android Inc.*", una pequeña empresa, la cual, posteriormente, fue comprada por Google, en la actualidad, Android es desarrollado por los miembros de la "*Open Handset Alliance"*, liderada por el propio Google.

Su presentación se realizó el 5 de noviembre de 2007 junto con la fundación "*Open Handset Alliance*", en un consorcio de numerosas compañías de hardware, software y telecomunicaciones comprometidas con la promoción de estándares abiertos para dispositivos móviles.

El SO Android es un sistema de código abierto, multitarea, que permite a los desarrolladores acceder a las funcionalidades principales del dispositivo mediante aplicaciones, cualquier aplicación puede ser reemplazada libremente, además, estas pueden ser desarrolladas por terceros, utilizando las herramientas proporcionadas por Google, mediante los lenguajes de programación Java y C.

El código fuente de Android está disponible bajo diversas licencias de software libre y código abierto, Google liberó la mayoría del código de Android bajo la licencia *Apache*. Todo esto, permite que un desarrollador no solo pueda modificar su código sino también mejorarlo. Una vez desarrolladas y probadas esas mejoras el desarrollador puede publicar el nuevo código y con él ayudar a mejorar el sistema operativo para futuras versiones.

Android depende de un núcleo (del inglés *kernel*) de Linux versión 2.6 para los servicios base del sistema como son la seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de drivers. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

Otras características que presenta Android son un navegador web integrado basado en el motor *WebKit*, soporte para gráficos 2D y 3D basado en la especificación "*Open Graphics Library*" (OpenGl), soporte multimedia para audio, video e imágenes en varios formatos, conectividad Bluetooth, *Enhanced Data Rates for "Global System for Mobile"* (GSM) *Evolution* (EDGE), tercera generación (3G) y WiFi entre otras.

2.1.2 Evolución de Android

A lo largo de los años, desde su lanzamiento, han surgido diversas versiones de Android. A modo de curiosidad, cabe destacar que todas ellas presentan nombres de postres.

Cuando queremos desarrollar una nueva aplicación Android es importante observar las evoluciones de las distintas versiones. Cada nueva versión de Android es compatible con las versiones anteriores pero esto no se cumple al revés. Cuando se lanza una nueva versión los cambios introducidos suelen ser aditivos, es decir, se añaden funcionalidades nuevas o se reemplazan las anteriores pero estas no se suelen borrar, las funcionalidades remplazadas se marcan como obsoletas (del inglés *deprecated*) pero no se borran, de esta forma las aplicaciones existentes las pueden seguir utilizando. En un número pequeño de casos, se pueden modificar o borrar algunas partes del sistema, estos cambios suelen llevarse a cabo para asegurar la robustez y la seguridad en la nueva versión de Android.

A continuación, se muestran las estadísticas de distribución de las diferentes versiones, en porcentaje, respecto al número total de dispositivos Android existentes. Estos datos hacen referencia a diciembre de 2012 y la fuente de estas estadísticas es Google. **[12]**

Versión	Nombre	API	Distribución
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Eclair	7	2.7%
2.2	Froyo	8	10.3%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	50.6%
3.1	Honeycomb	12	0.4%
3.2		13	1.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
4.1	Jelly Bean	16	5.9%
4.2		17	0.8%

Tabla 2.1. Distribución versiones Android [12]

Observando la **tabla 2.1** podemos concluir que la versión más extendida de Android es la llamada *Gingerbread*, esta representa casi el 51% del total, también, podemos observar que las versiones anteriores sólo suman en conjunto alrededor de un 14%, el resto de dispositivos que utilizan Android como SO utilizan versiones más recientes a la *Gingerbread*, siendo la versión 4.0 la que parece tomar el relevo a la versión 2.3. En la siguiente figura (**figura 2.1**) podemos observar estos mismos datos representados en un diagrama.

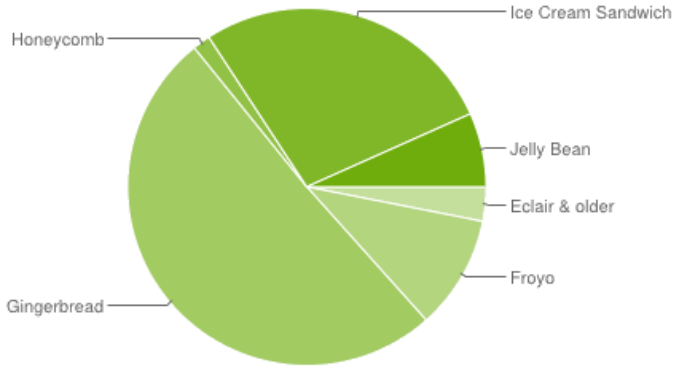


Figura 2.1. Diagrama distribución versiones Android

De la **figura 2.1** extraemos rápidamente que *Gingerbread* junto a "*Ice Cream Sandwich*" abarcan cerca del 75% de los dispositivos que usan el SO Android en la actualidad.

En la **figura 2.1** se muestra también que cada versión de Android tiene asignado un número entero que representa el nivel del marco de trabajo (del inglés *framework*) **API** (del inglés *Application Programming Interface*) que soporta esa versión. Cada versión de la plataforma Android soporta un único nivel del API, aunque implícitamente soporta

también todos los niveles anteriores. La primera versión utilizaba el nivel de API 1, según se crean nuevas versiones el número del API se incrementa.

Desde un punto de vista comercial, lo más interesante es que la aplicación sea compatible con el mayor número de dispositivos posibles, por lo que en ese caso, habría que elegir la versión de Android más baja posible sobre la que pueda correr nuestra aplicación. Por otra parte, también hay que tener en cuenta que una interfaz más atractiva aumentará el interés de los usuarios por la aplicación, esto lo podremos conseguir utilizando niveles superiores del API, una aplicación que sea compatible con todas las versiones de Android pero que no resulte atractiva no tendrá una gran difusión.

En nuestro caso, como se comentó en los objetivos del presente proyecto, nuestra plataforma tiene un fin divulgativo, es decir, está diseñada para llegar al mayor número posible de usuarios. La aplicación no es un fin en sí misma sino un medio para conseguir un objetivo, por lo tanto, el atractivo de la plataforma es conseguir ese objetivo para el que fue diseñada.

2.1.3 Arquitectura de Android

Antes de comenzar con el desarrollo de una aplicación Android, es importante conocer cómo está estructurado este sistema operativo, a esto le llamamos arquitectura. En el caso de Android está formada por varias capas que facilitan al desarrollador la creación de aplicaciones. Además, esta distribución permite acceder a las capas más bajas mediante el uso de librerías y, de esta forma, hacer que el desarrollador no tenga que programar a bajo nivel las funcionalidades necesarias para que una aplicación haga uso de los componentes hardware de los dispositivos.

Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, es por ello que a este tipo de arquitectura se le conoce también como pila. Para comprender mejor este aspecto, a continuación se muestra un diagrama que refleja la arquitectura del sistema Android (**figura 2.2**):

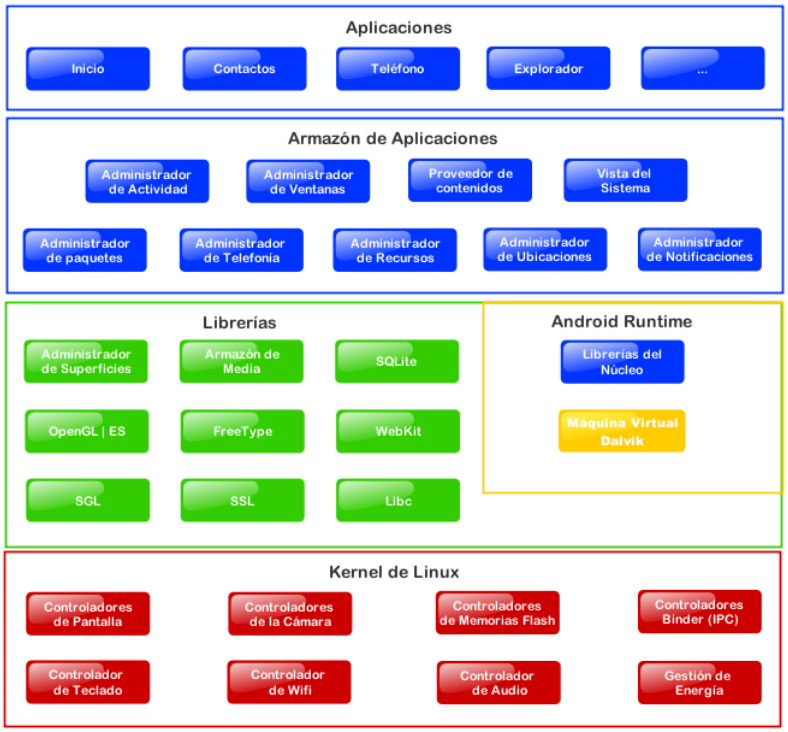


Figura 2.2. Arquitectura del sistema Android [13]

A continuación, describiremos un poco cada una de las capas mostradas en la **figura 2.2** siguiendo un orden ascendente **[13]**:

- **Núcleo de Linux**

  Como se comentó anteriormente, el núcleo del sistema operativo Android está basado en el núcleo de Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, con la diferencia de que está adaptado a las características del hardware sobre el que corre Android, en este caso, las características que ofrecen los dispositivos móviles.

  El núcleo actúa como capa de abstracción entre el hardware y el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. De esta forma, también se evita el hecho de que el desarrollador deba conocer las características concretas de cada dispositivo. Para cada elemento de hardware existe un controlador (del inglés *driver*) dentro del núcleo que permite controlarlo a través de una aplicación.

  Además, el núcleo se encarga de gestionar los diferentes recursos del dispositivo (energía, memoria, etc.) y de manejar el SO (procesos, elementos de comunicación, etc.).

- **Librerías**

Por encima del núcleo se encuentran las bibliotecas nativas de Android, también conocidas como librerías. Están escritas en C o C++ y compiladas para la arquitectura hardware específica de cada dispositivo. Normalmente, son desarrolladas por el propio fabricante del dispositivo, que además, se encarga de instalarlas en él. Las librerías tienen como objetivo proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando la máxima eficiencia respecto a los recursos del sistema.

Entre las librerías más comunes se encuentra OpenGL (encargado de generar los gráficos en la pantalla), bibliotecas multimedia (formatos de audio, imágenes y video), WebKit (navegador Web), *Secure Sockets Layer* (SSL), FreeType (fuentes de texto) y SQLite (base de datos) entre otras.

- **Entorno de ejecución**

El entorno de ejecución de Android, como muestra la **figura 2.3,** no se considera una capa en sí mismo, dado que también está formado por librerías. Podemos encontrar en él las librerías con las funcionalidades habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual ***Dalvik***. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y luego se pueden exportar a otros dispositivos donde pueden correr sin necesidad de ser recompiladas.

La máquina virtual *Dalvik* deriva de la máquina virtual de Java. Java se utiliza como base para desarrollar la aplicación, los ejecutables resultantes de la compilación obtenidos a través del SDK de Android tienen una extensión específica para *Dalvik* (la extensión es ***.dex***), por ello, no se puede correr una aplicación Java en el sistema Android ni una aplicación para Android puede correr en la máquina virtual de Java.

- **Entorno de trabajo de las aplicaciones**

La siguiente capa está compuesta por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual *Dalvik*. Podemos describir un poco cada una de ellas:

1. **Administrador de actividades** (del inglés *Activity Manager*): Se encarga de administrar la pila de actividades de una aplicación y también su ciclo de vida.

2. **Administrador de ventanas** (del inglés *Windows Manager*): Organiza lo que se mostrará en la pantalla del dispositivo. Crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.

3. **Proveedor de contenidos** (del inglés *Content Provider*): Esta librería crea una capa que encapsula los datos que se compartirán entre las aplicaciones para tener control sobre cómo se accede a dicha información.

4. **Vistas** (del inglés *Views*): Las vistas son los elementos que ayudan a construir las interfaces de usuario (botones, cuadros de texto, listas, etc.).

5. **Administrador de notificaciones** (del inglés *Notification Manager*): Engloba los servicios para notificar al usuario cuando se produzca un evento que requiera su atención mostrando alertas en la barra de estado. Esta biblioteca también permite manejar sonidos, activar el vibrador, etc.

6. **Administrador de paquetes** (del inglés *Package Manager*): Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Un paquete es la forma en que se distribuyen las aplicaciones Android (archivo ***.apk***), que a su vez, incluye los archivos ***.dex*** con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.

7. **Administrador de telefonía** (del inglés *Telephony Manager*): A través de esta librería podemos realizar llamadas o enviar y recibir mensajes de texto o multimedia, por el contrario, no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.

8. **Administrador de recursos** (del inglés *Resource Manager*): Gestiona todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o vistas.

9. **Administrador de localización** (del inglés *Location Manager*): Permite determinar la posición geográfica del dispositivo mediante GPS o redes disponibles y trabajar con mapas.

10. **Administrador de sensores** (del inglés *Sensor Manager*): Esta librería permite manipular los elementos de hardware del terminal como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, etc.

11. **Cámara**: A través de esta librería se puede utilizar la cámara, ya sea para tomar una fotografía o para grabar un video.

12. **Multimedia**: Permite reproducir audio, vídeo e imágenes en el dispositivo.

- **Aplicaciones**

  En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (desarrolladas en Java), las que se encuentran preinstaladas en el dispositivo y aquellas que instala el usuario.

  En esta capa también se encuentra la aplicación principal del sistema llamada "inicio" (del inglés *home*) o "lanzador" (del inglés *launcher*), esta aplicación permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso "***widgets***", que son también aplicaciones de esta capa.

2.1.4 Fundamentos de una aplicación Android

[14] Las aplicaciones Android están escritas en el lenguaje de programación Java. Las herramientas del *Software Development Kit* (SDK) de Android compilan el código y lo guardan en un paquete Android junto con los datos y los archivos de recursos, este paquete tiene forma de archivo con extensión "**.apk**". El código almacenado en este archivo se considera una aplicación y este es utilizado por los dispositivos Android para instalarla en su SO.

Una vez instalada la aplicación en el dispositivo, cada aplicación Android reside en su propia caja de arena (del inglés *sandbox*) de seguridad, algunas características de esa caja de arena son:

- El SO Android es un sistema Linux multiusuario en el cual cada aplicación es un usuario diferente.

- Por defecto, el sistema asigna a cada aplicación un identificador (ID) de usuario único, este ID sólo es usado por el sistema y es desconocido para la aplicación. El sistema otorga permisos para todos los archivos de una aplicación, de forma que sólo el usuario con la ID asignada a esa aplicación puede acceder a ellos.

- Cada proceso tiene su propia *virtual machine* (VM), así que el código de una aplicación se ejecuta de forma aislada al resto.

- Por defecto, cada aplicación corre en su propio proceso Linux. Android arranca el proceso cuando uno de los componentes de la aplicación necesita ser ejecutado, después, detiene el proceso cuando este ya no se necesita o cuando el sistema debe recuperar memoria para otras aplicaciones.

De este modo, el sistema Android implementa el principio de mínimos privilegios. Esto es, cada aplicación, por defecto, tiene acceso sólo a los componentes que requiere para hacer su trabajo y no más. Esto crea un entorno seguro en el cual una aplicación no puede acceder a partes del sistema para los cuales no se le han otorgado permisos.

Sin embargo, hay maneras para que una aplicación comparta datos con otras aplicaciones y para que una aplicación pueda acceder a servicios del sistema:

- Es posible configurar que dos aplicaciones compartan el mismo ID de usuario Linux, en este caso, cada una de las aplicaciones podría acceder a los archivos de la otra. Para conservar los recursos del sistema, las aplicaciones con el mismo ID de usuario pueden organizarse para correr en el mismo proceso Linux y compartir la misma VM, en ese caso las aplicaciones deben también estar firmadas con el mismo certificado de seguridad.

- Una aplicación puede solicitar permisos de acceso a datos del dispositivo como la lista de contactos del usuario, los *Short Message Service* (SMS), los dispositivos de almacenaje (como puede ser una tarjeta *Secure Digital* (SD)), cámara, Bluetooth… Todos los permisos de una aplicación tienen que ser concedidos por el usuario durante la instalación de esta.

2.1.4.1 Componentes de una aplicación

[14] Los componentes de una aplicación son los bloques esenciales de una aplicación Android. Cada componente es un punto diferente a través del cual el sistema puede entrar a la aplicación. No todos los componentes son puntos de entrada reales para el usuario y algunos son dependientes entre sí, no obstante, cada uno existe como su propia entidad y juega un papel específico, cada uno de ellos es una pieza única que ayuda a definir el comportamiento global de la aplicación.

Hay cuatro tipos de componentes de una aplicación. Cada tipo sirve para un determinado propósito y tiene un ciclo de vida diferente que define como se crea y se destruye el componente. Los cuatro tipos de componentes que existen son:

- **Actividades**

  Una actividad representa una pantalla simple con una interfaz de usuario. Por ejemplo, una aplicación que gestione el correo electrónico (conocidos como *emails*) debe tener una actividad que muestre una lista de mensajes recibidos, otra actividad que permita crear un nuevo email y otra actividad que permita leer emails. Aunque las actividades trabajan juntas para formar una experiencia cohesiva, desde el punto de vista del usuario en esta aplicación, cada una de las actividades depende de las otras. Como tal, una aplicación diferente puede lanzar cualquiera de esas actividades (si la aplicación descrita lo permite). Por ejemplo, una aplicación que maneje la cámara del dispositivo puede lanzar la actividad en la aplicación de correo electrónico que crea un nuevo email, de esta forma el usuario puede compartir una fotografía.

- **Servicios**

  Un servicio es un componente que corre en segundo plano para desempeñar operaciones de larga duración o realizar trabajos para procesos remotos. Un servicio no provee una interfaz de usuario. Por ejemplo, un servicio podría reproducir música en segundo plano mientras el usuario está en una aplicación diferente, o podría buscar datos sobre la red de trabajo sin bloquear la interacción del usuario con una actividad. Otro componente, tal como una actividad, puede lanzar el servicio y dejarlo corriendo o unirse a él con el fin de interactuar con él.

- **Proveedores de contenidos**

  Un proveedor de contenidos maneja un conjunto de datos compartidos de la aplicación. Los datos se pueden almacenar en un sistema de ficheros, en una base de datos SQLite, en la web, o en otro lugar de almacenamiento persistente donde la aplicación pueda acceder. Mediante el proveedor de contenidos, otra aplicación puede consultar o modificar los datos (si el proveedor de contenidos lo permite). Por ejemplo, el sistema Android ofrece un proveedor de contenidos que gestiona la información de los contactos del usuario. Como tal, una aplicación con los permisos adecuados puede consultar una parte del proveedor de contenidos para leer y escribir información sobre una persona en particular.

- **Receptores de difusión** (del inglés *Broadcast receivers*)

  Un receptor de difusión es un componente que responde a todo el sistema de avisos de difusión. Muchos de estos avisos se originan desde el sistema. Por ejemplo, un anuncio de difusión que avisa de que la pantalla se ha apagado, que la batería esta baja, o que una fotografía fue tomada. Las aplicaciones pueden también lanzar estos mensajes de difusión. Por ejemplo, para dejar que otra aplicación conozca que algunos datos han sido descargados del dispositivo y están disponibles para ser utilizados. Aunque los receptores de difusión no muestran una interfaz de usuario, ellos deben crear una barra de estado de notificaciones que alerte al usuario cuando un mensaje de difusión ocurre. Comúnmente, un receptor de difusión es sólo una puerta de entrada a otros componentes y está destinado a hacer una cantidad de trabajo muy pequeña. Por ejemplo, se podría iniciar un servicio para realizar un trabajo basado en el evento recibido.

Un aspecto único de diseño del sistema Android es que una aplicación puede lanzar un componente de otra aplicación. Por ejemplo, si queremos que el usuario capture una fotografía con la cámara del dispositivo, habrá probablemente otra aplicación que lo haga y esta aplicación puede usarla, en vez de desarrollar uno mismo una actividad que capture una foto. No se necesita ni incorporar ni enlazar el código de la aplicación que usa la cámara. En vez de eso, simplemente basta con lanzar la actividad de la aplicación que captura una foto. Cuando se completa, la fotografía se envía a la aplicación para que esta pueda usarla. De cara al usuario, esto parece como si la cámara fuera una parte de dicha aplicación.

Cuando el sistema lanza un componente, este comienza el proceso para esa aplicación (si este no está corriendo ya) e instancia las clases necesarias para el componente. Por ejemplo, si nuestra aplicación arranca la actividad que toma una fotografía en la aplicación que maneja la cámara, esa actividad corre en el proceso que lanzó la aplicación que maneja la cámara, no en el proceso de nuestra aplicación. Por lo tanto, a diferencia de la mayoría de las aplicaciones en otros sistemas, las aplicaciones Android no tienen solamente un punto de entrada (no hay un método *main*(), por ejemplo).

Debido a que el sistema corre cada aplicación en procesos independientes con archivos de permisos que restringen el acceso a otras aplicaciones, una aplicación no puede directamente activar un componente de otra aplicación. El sistema Android, sin embargo, si puede. Así, para activar un componente en otra aplicación, se debe lanzar un mensaje al sistema que especifique la intención de arrancar un componente en particular. El sistema activará el componente en nombre de la aplicación.

2.1.4.2 Activar componentes de una aplicación

[14] Tres de los cuatro tipos de componentes (actividades, servicios y receptores de difusión) se activan mediante un mensaje asíncrono llamado **intento** (del inglés *intent*). Los intentos enlazan componentes individuales con otros en tiempo de ejecución (se puede pensar en ellos como los mensajeros que solicitan una acción a otro componente), tanto si los componentes pertenecen a nuestra aplicación como si pertenecen a otra.

Un intento es creado con un objeto *Intent*, el cual, define un mensaje para activar uno de los otros componentes específicos o un tipo especifico de componente, un intento puede ser tanto explícito como implícito, respectivamente.

Para actividades o servicios, un intento define la acción a realizar (por ejemplo, ver o enviar algo) y debe especificar el *Uniform Resource Identifier* (URI) de los datos sobre los que actuar (además de otras cosas que el componente necesite saber para ser lanzado). Por ejemplo, un intento puede transmitir una solicitud para una actividad que muestre una imagen o que abra una página web. En algunos casos, puede arrancar una actividad para recibir un resultado, en cuyo caso, la actividad también devolvería el resultado en un intento (por ejemplo, se puede emitir un intento de permitir al usuario elegir un contacto personal y que lo devuelva, el intento devuelto incluirá una URI que apunte al contacto elegido).

Para los receptores de difusión, el intento simplemente define el anuncio que será difundido (por ejemplo, un mensaje de difusión para indicar que la batería del dispositivo está baja sólo incluye un mensaje de acción conocido que dice "*battery is low*").

El otro tipo de componente, el proveedor de contenidos, no se activa por intentos. Más bien, es activado cuando recibe una solicitud de un objeto *ContentResolver*. Este objeto maneja todas las transacciones directas con el proveedor de contenidos así que el componente que está realizando transacciones con el proveedor no necesita intentos, en vez de eso, llama a los métodos propios del objeto *ContentResolver*. Esto deja una capa de abstracción entre el proveedor de contenido y la información del componente que lo solicita (por seguridad).

2.1.4.3 El archivo manifiesto

[14] Antes de que el sistema Android pueda lanzar un componente de una aplicación, el sistema debe conocer la existencia de ese componente leyendo el archivo *AndroidManifest.xml* de la aplicación (el archivo manifiesto, del inglés *manifest file*). Una aplicación debe declarar todos sus componentes en este archivo, el cual, debe estar en el directorio raíz del proyecto que contiene la aplicación.

El archivo manifiesto además de declarar los componentes de la aplicación tiene otra serie de cometidos, como son:

- Identificar algunos de los permisos de usuario que la aplicación requiere.
- Declarar el nivel mínimo del API requerido por la aplicación, basado en los API que usa la aplicación.
- Declarar el hardware y el software del dispositivo que usa o requiere la aplicación, como puede ser la cámara, el Bluetooth, etc.
- Las librerías API con las que la aplicación necesita enlazarse (otras distintas a las que ofrecen los API del entorno de trabajo Android), como puede ser la librería de "*Google Maps*".

2.1.4.4 Capacidades de los componentes

[14] Como ya hemos visto, podemos usar los intentos para comenzar actividades, servicios y receptores de eventos de difusión. Esto se puede hacer llamando explícitamente al componente destino (usando el nombre de clase del componente) en el intento. Sin embargo, el poder real de los intentos reside en el concepto de acciones de intento. Con las acciones de intento, simplemente hay que indicar el tipo de acción que queremos realizar (y opcionalmente, los datos sobre los cuales queremos realizar la acción) y dejar que el sistema encuentre un componente en el dispositivo que pueda llevar a cabo la acción y que la lance. Si existen varios componentes que pueden llevar a cabo la acción descrita en el intento, será el usuario el que decida cuál de ellos usar.

La manera en la que el sistema identifica los componentes que pueden responder al intento es comparando el intento recibido con los filtros de intentos declarados en los archivos manifiestos de las otras aplicaciones instaladas en el dispositivo.

Cuando declaramos un componente en el archivo manifiesto de nuestra aplicación, podemos opcionalmente incluir filtros de intentos que indiquen las capacidades del componente para que pueda responder a intentos recibidos desde otras aplicaciones.

2.1.4.5 Requerimientos de la aplicación

[14] Hay una gran variedad de dispositivos que poseen el SO Android y no todos ellos ofrecen las mismas capacidades y características. Para evitar que nuestra aplicación se instale en dispositivos que no poseen las características necesarias, es importante definir el perfil de tipos de dispositivos que la aplicación soporta declarando los requisitos en el archivo manifiesto. Muchas de estas declaraciones son meramente informativas y el sistema no las lee, por el contrario, los servicios externos como "*Google Play*" las leen para llevar a cabo filtros que los usuarios puedan utilizar cuando buscan aplicaciones para su dispositivo. Estos filtros evitan que los usuarios instalen aplicaciones incompatibles con sus aparatos.

Se puede, sin embargo, declarar que una aplicación usa la cámara, pero que no la requiere obligatoriamente. En este caso, la aplicación debe realizar un chequeo en tiempo de ejecución para comprobar si el dispositivo tiene una cámara y desactivar las características de la aplicación que usan la cámara si esta no está disponible.

Aquí presentamos algunas características importantes del dispositivo que deben ser consideradas cuando diseñamos y desarrollamos una aplicación:

- **Tamaño de la pantalla y su densidad**

  Android define varios tamaños de pantalla y varias densidades de estas. La densidad de una pantalla se mide en píxeles por pulgada. Por defecto, una aplicación es compatible con todos los tamaños y densidades, ya que el sistema Android hace los ajustes necesarios. Sin embargo, se pueden definir diferentes disposiciones de pantalla según el tamaño y la densidad de esta, o incluso utilizar imágenes de cierto tamaño y densidad en cada caso concreto.

- **Configuraciones de entrada**

  Algunos dispositivos proveen diferentes tipos de mecanismos de entrada de datos del usuario, como puede ser un teclado externo, un ratón, etc. Si la aplicación requiere un tipo especifico de hardware de entrada, se debe declarar en el archivo manifiesto.

- **Características del dispositivo**

  Hay muchas características de hardware y software que pueden no existir o no estar disponibles en un dispositivo Android, como la cámara, un sensor de luz, Bluetooth o una cierta versión de OpenGl. Nunca se debe dar por supuesto que una característica esté disponible en todos los dispositivos Android (siempre que sea distinta de las librerías estándar de Android), así pues, se deben declarar las características que usa la aplicación en el archivo manifiesto.

- **Versión de la plataforma**

  Dispositivos diferentes a menudo utilizan versiones de Android diferentes. Si la aplicación hace uso de algún API no incluido en el nivel 1, se debe declarar el nivel mínimo del API necesario para que corra la plataforma.

Es importante declarar cada uno de los requerimientos de la aplicación, ya que cuando esta se distribuya en "*Google Play*", el almacenador usará estas declaraciones para filtrar que aplicaciones son compatibles con cada dispositivo. Así pues, la plataforma sólo debería estar disponible para dispositivos que dispongan de los requerimientos necesarios.

2.1.4.6 Recursos de la aplicación

[14] Una aplicación Android está compuesta además del código fuente por otros elementos, una aplicación requiere recursos que están separados del código fuente,

como son imágenes, ficheros de audio y todo lo relativo a la presentación visual de la aplicación. Por ejemplo, se pueden definir animaciones, menús, estilos, colores, y las vistas de las actividades en archivos de tipo *eXtensible Markup Language* (XML). Usando recursos de aplicaciones es más sencillo modificar ciertas características de la aplicación sin modificar el código y, además, si se proveen conjuntos de recursos alternativos se puede optimizar la aplicación para varias configuraciones distintas de dispositivos, como pueden ser diferentes lenguajes o tamaños de pantalla.

Para cualquier recurso que se incluya en un proyecto Android, las herramientas del constructor SDK definen un entero único ID, el cual se puede usar para referenciar el recurso desde el código de la aplicación o desde otros recursos definidos en un XML.

Uno de los aspectos más importantes de proveer recursos separados del código fuente es la capacidad para proporcionar diferentes recursos para distintas configuraciones de dispositivos. Por ejemplo, se pueden definir las etiquetas de la aplicación en varios idiomas guardándolas en distintos archivos XML, después, teniendo en cuenta los lenguajes proporcionados por dicha aplicación y la configuración del idioma definida en el dispositivo del usuario, se establecerá el idioma a mostrar.

Android soporta muchos calificadores diferentes para los recursos alternativos. El calificador es una etiqueta corta que incluye el nombre del directorio de recursos para definir la configuración del dispositivo para el cual esos recursos deberían ser usados. Como ejemplo, podemos definir varias vistas para una misma actividad, dependiendo del tamaño de la pantalla y la orientación del dispositivo. En el caso de la orientación de la pantalla se pueden definir dos vistas diferentes y aplicar un calificador apropiado para cada nombre de directorio de vistas. Después, el sistema automáticamente aplica la vista apropiada dependiendo de la orientación del dispositivo en ese momento.

2.1.5 APIs de Android destacados en este proyecto

En las sucesivas secciones se van a describir los API más importantes utilizados para la realización del presente proyecto.

2.1.5.1 Actividades

[14] El componente básico y el único utilizado de los que provee Android para el desarrollo de la aplicación resultante del presente proyecto son las actividades.

Una actividad es un componente de aplicación que provee de una pantalla con la cual el usuario puede interactuar con objeto de que este pueda hacer algo, como puede ser el dial del teléfono, tomar una foto, enviar un email, o ver un mapa. Cada actividad proporciona una ventana en la cual se dibuja su interfaz de usuario. Dicha ventana típicamente llena la pantalla, pero puede ser más pequeña que la pantalla y flotar encima de otras ventanas.

Una aplicación, normalmente, consiste en múltiples actividades que están débilmente unidas entre ellas. Típicamente, una actividad en una aplicación se especifica como la actividad principal, la cual se presenta al usuario cuando se lanza la aplicación. Cada actividad puede posteriormente arrancar otras actividades para realizar diferentes acciones. Cada vez que se lanza una nueva actividad, la actividad previa se detiene pero el sistema conserva la actividad en una pila (llamada pila trasera, del inglés *back stack*). Cuando una actividad comienza, esta se inserta en la pila trasera y toma el foco del usuario. La pila trasera se basa en el mecanismo de pila consistente en que la última en entrar es la primera que sale (del inglés *last in, first out*), así, cuando el usuario ha terminado con la actividad actual y presiona el botón volver, esta actividad desaparece de la pila y la actividad previa se reanuda.

Cuando una actividad es detenida porque una nueva actividad comienza, se notifica el cambio de estado a través de los métodos de devolución de llamadas (del inglés *callback*) del ciclo de vida de la actividad. Hay varios métodos de devolución de llamadas que una actividad podría recibir, debido a un cambio es su estado (si el sistema la crea, la detiene, la reanuda o la destruye), cada devolución de llamada proporciona la oportunidad de realizar un trabajo especifico que sea apropiado al cambio de estado. Por ejemplo, cuando una actividad es detenida, dicha actividad debería liberar algunos objetos grandes, como una conexión a una red o a una base de datos. Cuando la actividad se reanuda, se puede obtener de nuevo los recursos necesarios y reanudar las acciones que fueron interrumpidas. Todas estas transiciones en los estados forman parte del ciclo de vida de una actividad.

Manejar correctamente el ciclo de vida de las actividades implementando los métodos de devolución de llamadas es crucial para desarrollar una aplicación robusta y a la vez flexible. El ciclo de vida de una actividad está afectado directamente por sus asociaciones con otras actividades, su tarea y la pila trasera.

Una aplicación puede existir esencialmente en tres estados:

- **Reanudada** (del inglés *resumed*)

  La actividad está en primer plano de la pantalla y tiene el foco de usuario. A este estado también se le conoce como “corriendo” (del inglés *running*).

- **Pausada** (del inglés *paused*)

  Otra actividad se encuentra en primer plano y tiene el foco del usuario, pero esta actividad es todavía visible. Es decir, otra actividad es visible encima de esta y esta actividad es parcialmente transparente o no cubre toda la pantalla. Una actividad pausada está completamente viva (el objeto *Activity* está almacenado en la memoria, se mantiene todo el estado y la información de los miembros, además, esta permanece unida al gestor de ventanas) pero puede ser

matada (del inglés *killed*) por el sistema en caso de tener poca memoria disponible.

- **Detenida** (del inglés *stopped*)

  La actividad se encuentra en segundo plano, no es visible en absoluto ya que otra actividad ocupa la pantalla. Una actividad detenida esta también viva (el objeto *Activity* está guardado en memoria, se mantiene el estado y la información de los miembros, pero no está unida al gestor de ventanas). Sin embargo, no es visible para el usuario y puede ser matada por el sistema si este necesita la memoria para otra cosa.

Si una actividad esta pausada o detenida, el sistema puede sacarla de la memoria ya sea finalizándola (llamando al método *finish*()), o simplemente matando su proceso. Cuando una actividad es abierta de nuevo (después de ser finalizada o matada), debe ser creada otra vez completamente.

2.1.5.2 Intentos

[**14**] Tres de los componentes del núcleo de una aplicación (actividades, servicios y receptores de difusión) son activados a través de mensajes, llamados intentos (del inglés *intents*). El envío de intentos tiene como objeto enlazar componentes en tiempo de ejecución, ya sean de la misma aplicación o de aplicaciones diferentes. El intento es un objeto *Intent*, es una estructura de datos pasiva que contiene una descripción abstracta de la operación que será realizada. En el caso de los eventos de difusión, una descripción de algo que ha sucedido y está siendo anunciado. Existen mecanismos independientes para la entrega de intentos para cada tipo de componente.

- Un objeto *Intent* es pasado al método *Context.startActivity*() o al método *Activity.startActivityForResult*() para lanzar una actividad o hacer que una actividad ya creada haga algo nuevo.

- Un objeto *Intent* es pasado al método *Context.startService*() para iniciar un servicio o entregar instrucciones nuevas a un servicio que se ejecuta de forma continua. De manera similar, un intento puede ser pasado al método *Context.bindService*() para establecer una conexión entre el componente llamado y el servicio elegido. Esto puede, opcionalmente, iniciar el servicio si este no está corriendo en ese instante.

- Un objeto *Intent* pasado a cualquiera de los métodos de difusión es enviado a todos los receptores de difusión que procedan en cada caso. Muchos tipos de eventos de difusión son originados en el código del sistema.

El sistema Android encuentra el componente apropiado en cada caso, instanciándolo si es necesario. No existe solapamiento entre estos sistemas de mensajería. Los intentos de difusión son entregados sólo a receptores de difusión, nunca se entregan a servicios o actividades. Un intento pasado al método *startActivity*() es entregado sólo a una actividad, nunca a un servicio o a un receptor de difusión, etc.

Un objeto *Intent* es un conjunto de información. Este objeto contiene información interesante para el componente que lo recibe, además de información de interés sobre el sistema Android. Principalmente contiene:

- Nombre del componente.
- Acción a realizar.
- La URI de los datos sobre los que actuar y el tipo *Multipurpose Internet Mail Extensions* (MIME) de estos.
- Información adicional sobre el tipo de componente.
- Datos adicionales que el componente espera encontrar dentro del intento.
- Algunas banderas.

En nuestra aplicación, las diferentes actividades que la componen se comunican entre sí por medio de estos objetos.

2.1.5.3 Servicios de localización

Para poder cumplir los objetivos para los que fue diseñada la aplicación desarrollada en este proyecto necesita recibir datos de localización del GPS del dispositivo. Con objeto de recibir y procesar dichos datos son necesarios los APIs de Android que se encargan de manejar las capacidades de localización del aparato.

[14] Android proporciona acceso a los servicios de localización soportados por el dispositivo a través de las clases incluidas en el paquete *android.location*. El componente central del entorno de trabajo de la localización es el servicio de sistema *LocationManager*, el cual, provee APIs para determinar la localización y el rumbo del dispositivo subyacente (si está disponible).

Así como otros servicios del sistema, un objeto de la clase *LocationManager* no se instancia directamente. En vez de eso, se solicita una instancia al sistema y este devuelve un manejador de dicha clase.

Una vez que la aplicación tiene un manejador de *LocationManager*, la aplicación está lista para hacer tres cosas:

- Consultar a la lista de todos los *LocationProvider* para obtener la última localización conocida del usuario.
- Registrar o quitar del registro de un proveedor de localización al usuario para que este reciba actualizaciones periódicas de su localización actual.

- Registrar o quitar del registro un intento dado si está contenido dentro de una proximidad determinada definida por el conjunto de coordenadas formado por latitud y longitud.

A la hora de desarrollar una aplicación que utilice la ubicación del dispositivo, se puede utilizar GPS y el proveedor de localización de red de Android para determinar la localización del usuario. Cada una de las dos opciones tiene sus ventajas y sus inconvenientes. GPS es más preciso, trabaja al aire libre (no funciona dentro de edificios), consume más batería y tarda en devolver la primera localización. Por su parte, el proveedor de localización de red de Android determina la posición usando las células de las antenas de telefonía (Cell-ID) y las señales WiFi, por lo que funciona tanto dentro de los edificios como fuera, es más rápido y consume menos batería. Para obtener la ubicación del usuario en una aplicación, se pueden usar ambos métodos a la vez o sólo uno de ellos.

Obtener la localización del usuario en un dispositivo móvil puede ser complicado. Hay varias razones por las que una lectura de posición puede contener errores y puede ser imprecisa. Algunas fuentes de error en la localización del usuario son:

- **Múltiples fuentes de localización**

  GPS, Cell-ID y el WiFi pueden cada uno proveer una pista de la ubicación del usuario. La determinación de cual utilizar es una cuestión de encontrar el equilibrio necesario entre la precisión, la velocidad y la eficiencia de consumo de la batería.

- **El movimiento del usuario**

  Ya que la localización del usuario cambia, se debe establecer la frecuencia de actualización de la posición del usuario adecuadamente.

- **Variaciones en la precisión**

  Las estimaciones de la ubicación del usuario procedentes de diferentes fuentes no son consistentes en cuando a su precisión. Por ejemplo, una posición obtenida diez segundos antes de una determinada fuente podría ser más precisa que una nueva ubicación obtenida de la misma fuente o de otra.

Teniendo en cuenta el uso que se le da a la posición actual del usuario en nuestra aplicación (determinar velocidad, aceleración, etc), el error en la posición que obtenemos utilizando el proveedor de localización de red de Android se aleja mucho de los mínimos tolerables. En la práctica se observaron márgenes de error en las coordenadas de 100 metros.

2.1.5.4 Interfaz de usuario

[14] Todos los elementos de la interfaz de usuario en una aplicación Android son construidos usando el objeto *View* y el objeto *ViewGroup*. *View* es un objeto que dibuja algo en la pantalla con lo que el usuario puede interactuar. Un *ViewGroup* es un objeto que contiene otros objetos *View* (y *ViewGroup*) con el fin de definir el diseño de la interfaz.

La interfaz de usuario para cada componente de una aplicación es definida usando una jerarquía de objetos *View* y *ViewGroup*, como se muestra en la **figura 2.3**. Cada grupo de vistas (del inglés *ViewGroup*) es un contenedor invisible que organiza las vistas hijas, mientras, las vistas hijas pueden ser controles de entrada u otros *widgets* que dibujen alguna parte de la interfaz de usuario. Este árbol jerárquico puede ser tan simple o tan complejo como el diseñador necesite que sea.

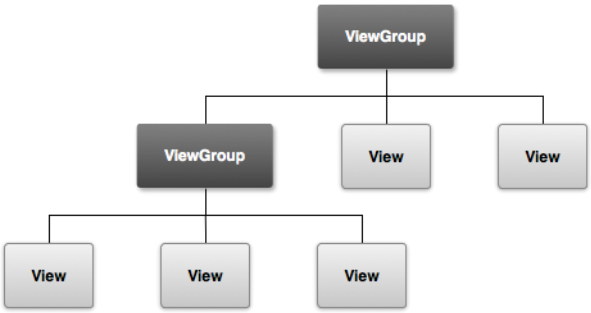


Figura 2.3. Árbol jerárquico que define el diseño de una interfaz

Para declarar un diseño, se pueden instanciar objetos *View* en el código y comenzar a construir el árbol, pero, la forma más sencilla y efectiva de definir un diseño es mediante un archivo XML. XML ofrece una estructura legible, similar a *HyperText Markup Language* (HTML), para el desarrollador del diseño.

2.1.5.5 Menús

[14] Los menús son un componente de interfaz de usuario común en muchos tipos de aplicaciones. Para proveer una experiencia al usuario familiar y consistente, se deben usar las APIs *Menu* para presentar acciones de usuario y otras opciones en nuestras actividades.

A partir de Android 3.0 (API nivel 11), los dispositivos Android no requieren de un botón dedicado al menú. Con este cambio, las aplicaciones Android deben dejar atrás la dependencia al panel de 6 elementos tradicional y proporcionar en su lugar una barra de acción (del inglés *action bar*) para presentar las acciones comunes de los usuarios.

Aunque el diseño y la experiencia del usuario para algunos elementos del menú ha cambiado, la semántica para definir un conjunto de acciones y opciones todavía se basa en los APIs *Menu*. Los tipos fundamentales de menús son:

- **Menú de opciones y barra de acciones**

  El menú de opciones es la colección primaria de elementos de menú para una actividad. Es decir, donde se deben emplazar las acciones que tienen un impacto global en la aplicación.

  Si se desarrolla para Android 2.3 o anterior, los usuarios pueden ver el panel del menú de opciones presionando el botón "*Menú*" de su dispositivo.

  En Android 3.0 y posteriores, los elementos del menú de opciones se encuentran en la barra de acción como una combinación de elementos de acción mostrados en la pantalla y otros ocultos. Comenzando con Android 3.0, el botón "*Menú*" está obsoleto (algunos dispositivos incluso no tienen dicho botón) por lo que se debe migrar hacia el uso de la barra de acciones para facilitar el acceso a las acciones y otras opciones.

- **Menú de contexto y modo de acción contextual**

  Un menú de contexto es un menú flotante que aparece en la pantalla cuando el usuario realiza una pulsación larga en un elemento. Esto provee acciones que afectan al contenido seleccionado o al marco del contexto.

  Cuando se desarrolla para Android 3.0 y posteriores, se debe utilizar, en vez del menú de contexto, el modo de acción contextual para activar acciones o seleccionar contenido. Este modo muestra elementos de acción que afectan al contenido seleccionado en una barra en la parte superior de la pantalla y permite al usuario seleccionar múltiples elementos.

- **Menú emergente** (del inglés *popup*)

  Un menú emergente muestra una lista de elementos en una lista vertical que está anclada a la vista que invoca el menú. Es una buena elección para proveer una serie de acciones que no caben en la pantalla y que están relacionadas con el contenido específico, o bien, para proveer opciones para la segunda parte de un comando. Las acciones en un menú emergente no deberían de afectar directamente al contenido correspondiente, de eso se encargan las acciones contextuales. Por el contrario, el menú emergente es para las acciones extendidas que se relacionan con regiones de contenido de la actividad.

Para todos los tipos de menú, Android provee un formato estándar XML para definir los elementos de un menú. En vez de construir un menú en el código de la actividad, se debe definir el menú y todos sus elementos en un archivo XML. Después de crear dicho archivo, se puede inflar el recurso de menú (cargarlo como un objeto *Menu*) en la actividad o en un fragmento de esta.

2.1.5.6 Diálogos

[14] Un diálogo es una ventana pequeña que indica al usuario que ha de tomar una decisión o introducir información adicional. Los diálogos no ocupan toda la pantalla y se utilizan, normalmente, para aquellos eventos que requieren al usuario para realizar una acción antes de que estos eventos puedan llevarse a cabo.

La clase básica para usar diálogos es la clase *Dialog*. Se debe evitar crear instancias de esta clase directamente, en su lugar, hay que utilizar una de las siguientes subclases:

- **AlertDialog**: Esta clase crea un dialogo que puede mostrar un título, un máximo de tres botones, una lista de elementos o una vista.

- ***DatePickerDialog*** o ***TimePickerDialog***: Un diálogo con una interfaz de usuario predefinida que permite al usuario seleccionar una fecha o una hora.

Los diálogos han sido utilizados, en la plataforma diseñada en este proyecto, en dos tareas. La primera de ellas con objeto de que el usuario confirme la configuración inicial de la aplicación que usará esta como base de su funcionamiento. La segunda tarea que realizan los diálogos en el presente proyecto es mostrar indicaciones sobre el funcionamiento de la aplicación cuando el usuario selecciona en el menú la opción “ayuda”.

2.1.5.7 Tostadas

[14] Una tostada (del inglés *Toast*) provee una reacción a una operación en una pequeña pantalla emergente. Sólo ocupa el espacio necesario para mostrar el mensaje que contiene y permite que la actividad que se encuentra en primer plano permanezca visible y siga siendo interactiva. Las tostadas desaparecen después de un tiempo de espera.

Por ejemplo, si el usuario se encuentra escribiendo un correo electrónico y antes de enviarlo decide navegar por su contenedor de correos electrónicos (buscando, por ejemplo, algún dato en otro email enviado o recibido) provocará que se muestre un mensaje indicando que el email que estaba editando se ha guardado como un borrador, de esta forma, el usuario sabe que puede reanudar posteriormente la edición del mismo sin perder información.

Estos elementos de Android se han utilizado para informar de dos eventos. El primero de ellos informa acerca de la disponibilidad del GPS instalado en el terminal, de modo que informará al usuario si el GPS se encuentra desactivado. En segundo lugar, este componente informa al usuario de que los efectos sonoros de la aplicación han sido activados o desactivados.

2.1.5.8 Barras de búsqueda de progreso

[14] Una barra de progreso (del inglés *ProgressBar*) es un indicador visual de la evolución de alguna operación. Este componente muestra una barra al usuario representando en qué medida la operación que se está realizando ha avanzado. La aplicación puede cambiar la cantidad de progreso a medida que avanza modificando la longitud de la barra.

Se puede mostrar un progreso secundario en la barra de progreso que será útil para mostrar un progreso intermedio, tal como el uso del espacio de memoria reservado en una barra de progreso que muestre una reproducción de una transmisión multimedia.

Una barra de progreso puede ser construida como indeterminada. Una barra indeterminada muestra una animación cíclica sin indicación de progreso. Este tipo de barra se utiliza para aplicaciones de las cuales desconocemos, a priori, la longitud de la tarea a realizar. Una barra de progreso indeterminada se puede configurar como una rueda que gira o una barra horizontal.

Una barra de búsqueda de progreso (del inglés *SeekBar*) es una extensión de una barra de progreso que añade un indicador arrastrable. El usuario puede tocar el indicador y arrastrarlo a izquierda o derecha para ajustar el nivel de progreso actual, o bien, utilizar las teclas de las flechas. El rango de la barra de búsqueda abarca desde 0 hasta el valor entero máximo configurado.

En este proyecto se han utilizado barras de búsqueda de progreso para que el usuario configure algunos valores numéricos dentro de los rangos establecidos. De esta forma, se consigue evitar tener código que compruebe los valores introducidos por el usuario, lo que ocurriría si utilizásemos etiquetas de texto normales.

2.1.5.9 Reproducción de medios

El marco de trabajo multimedia de Android incluye soporte para reproducir los tipos más comunes de estos archivos, de esta manera se puede integrar fácilmente audio, vídeo y también imágenes en las aplicaciones. Permite reproducir pistas de audio o vídeo desde archivos de medios guardados en los recursos de la aplicación, desde archivos guardados en el sistema de ficheros o desde un flujo de datos que llega desde una conexión de red. Para todos ellos, podemos usar los APIs llamados *MediaPlayer*.

Las siguientes clases se usan para reproducir sonido y vídeo en el entorno de trabajo de Android:

- ***MediaPlayer***: Esta clase es el API primario para reproducir sonido y vídeo.

- ***AudioManager***: Esta clase gestiona las fuentes de audio y la salida de audio en un dispositivo.

Con objeto de dotar a nuestra aplicación de efectos de sonido, se ha utilizado la clase *SoundPool* que se encarga de gestionar y reproducir recursos de audio en aplicaciones, para su configuración se ha establecido como tipo de flujo de datos el tipo *AudioManager.STREAM_MUSIC*.

Un objeto *SoundPool* es una colección de muestras que pueden ser cargadas en memoria desde un recurso dentro del archivo de la aplicación o desde un archivo que se encuentre en el sistema de ficheros. La librería *SoundPool* utiliza el servicio *MediaPlayer* para decodificar el audio en un flujo de datos sencillo. Esto permite a las aplicaciones evitar la latencia y la carga de la *Central Processing Unit* (CPU) que se produciría al descomprimir los archivos durante la reproducción.

Además de reproducir con baja latencia, *SoundPool* puede gestionar varias pistas de audio a la vez. Cuando el objeto *SoundPool* está construido, hay un parámetro que fija el número máximo de flujos de audio que pueden ser reproducidos a la vez desde un sólo objeto *SoundPool*. Si el número máximo de flujos de audio es excedido, automáticamente detiene una reproducción anterior basándose primero en la prioridad y después en la edad dentro de esa prioridad. Limitando el número máximo de flujos de audio ayudamos a no sobrecargar la CPU del dispositivo cargando las pistas y reducimos la probabilidad de que los flujos de audio se mezclen, lo que podría influir en la comprensión o en el rendimiento de la interfaz de usuario.

2.2 JAVA

En 1991 “Sun Microsystems” desarrolló el lenguaje de programación orientado a objetos que se conoce como Java. La intención de Sun fue crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel.

Algunas características de Java son **[15]**:

- **Es un lenguaje simple**: Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir aplicaciones web (conocidas como applets) interesantes desde el principio. Todos los desarrolladores familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de los desarrolladores los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

- **Es orientado a objetos**: Desde el principio Java fue diseñado como un lenguaje orientado a objetos. Los objetos se agrupan en estructuras encapsuladas, tanto sus datos como los métodos (o funciones) que manipulan esos datos. La

tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, sobre todo en entornos cada vez más complejos y basados en red.

- **Distribuido**: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir enchufes (del inglés *sockets*) y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

- **Interpretado y compilado a la vez**: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los *bytecodes*, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (del inglés *run-time*).

- **Robusto**: Java fue diseñado para crear software altamente fiable. Para ello, proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

- **Seguro**: Dada la naturaleza distribuida de Java, donde las *applets* se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su ordenador programas procedentes de fuentes desconocidas con acceso total a su sistema. Así, se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

- **Indiferente a la arquitectura**: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan variopintos, el compilador de Java genera *bytecodes*. Los *bytecodes* son un formato intermedio, indiferente a la arquitectura, diseñado para transportar el código de manera eficiente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

- **Portable**: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Esta última característica junto con la anterior forman lo que se conoce como la **Maquina Virtual Java** (**JVM**).

- **Multihilo**: Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (del inglés *multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

- **Dinámico**: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la red.

- **Produce *applets***: Java puede ser usado para crear dos tipos de programas: aplicaciones y *applets*. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, por ejemplo, el navegador web *HotJava*, escrito íntegramente en Java. Por su parte, las *applets* son pequeños programas que aparecen embebidos en las páginas web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

2.3 Conducción eficiente

[10] Como se comentó en la introducción de la presente memoria, la "conducción eficiente" se rige por unas reglas sencillas y eficaces que tratan de aprovechar las posibilidades que ofrecen las tecnologías de los motores de los coches actuales.

Las ventajas y objetivos que ofrece y persigue este tipo de conducción afectan al conductor, al vehículo y en general al entorno de ambos. Enumeraremos a continuación los más importantes:

- Mejorar el confort de la conducción reduciendo a la vez la tensión en el conductor.
- Reducir el riesgo de sufrir un accidente y, en caso de sufrirlo, minimizar la gravedad de este.
- Ahorro del gasto en combustible.
- Reducir los costes de mantenimiento del vehículo (frenos, embrague, caja de cambios, neumáticos y motor).
- Reducción de la contaminación urbana mejorando así la calidad del aire.
- Reducción de emisiones de CO_2 y, así, ayudar a cumplir los acuerdos internacionales en esta materia que tiene como fin último disminuir el problema del calentamiento global (efecto invernadero).

- Ahorrar energía a escala nacional incidiendo en la balanza de pagos y reduciendo la dependencia energética exterior (en el caso de España).

2.3.1 El motor

[10] El volumen de combustible que utiliza el motor en cada instante depende directamente de la potencia demandada sobre este, además, hay que tener en cuenta que el consumo del motor también varía en función de la temperatura, cuando el motor está frío consume más que cuando alcanza su temperatura óptima de funcionamiento. Con el motor en su temperatura ideal la potencia demandada sobre este depende de dos factores: la posición del pedal del acelerador y el régimen de revoluciones del motor. Estos dos factores son las condiciones impuestas por el conductor, de la posición del pedal y de la marcha engranada depende el consumo real del vehículo.

Para entregar una determinada potencia o rodar a cierta velocidad existen varias posibles combinaciones entre la posición del pedal y la marcha engranada en la caja de cambios. Para una misma velocidad, utilizar la marcha más larga posible hace que el motor gire a menos revoluciones y, por lo tanto, consuma menos.

También se consume más carburante cuando se demanda más potencia del motor. Se demanda menos potencia del motor cuando se utilizan aceleraciones más pequeñas (se aumenta la velocidad más lentamente), cuando se circula a menor velocidad o cuando circulamos por una pendiente descendente.

2.3.2 Eficiencia del combustible

[10] El carburante (ya sea gasolina o gasóleo) libera energía térmica a través de la combustión dentro de los cilindros del motor. Esta energía se transforma en trabajo mecánico proporcionando el movimiento a las ruedas del vehículo. En el mejor de los casos, de la energía que libera el combustible sólo se podría aprovechar el 38%, este porcentaje se reduce considerablemente cuando se circula por ciudades, ya que en ellas son muy frecuentes los arranques y las paradas. Uno de los objetivos de la “conducción eficiente” es aumentar el rendimiento del carburante consumido.

De la energía contenida en un litro de gasolina, el 62% se pierde por fricción y calor en el motor. En conducción urbana se pierde un 17% por marcha en vacío o ralentí a causa del tiempo que se pierde en las paradas con el motor encendido. El 21% restante de la energía llega al embrague. Las pérdidas producidas por la transmisión son del 6%. El resultado es que sólo el 15% de la energía se utiliza para mover el vehículo. Podemos observar estos datos en la **figura 2.4**.

Figura 2.4. Distribución energía consumida [10]

2.3.3 Reglas de la conducción eficiente

[10] Con el fin de optimizar la conducción de un vehículo a motor, las principales claves de la conducción eficiente son:

- Circular en la marcha más baja posible y a bajas revoluciones.
- Mantener la velocidad de circulación lo más uniforme posible.
- En los procesos de aceleración, cambiar de marcha entre 2.000 y 2.500 rpm en los motores de gasolina, o bien, entre 1.500 y 2.000 rpm en los motores diesel.
- En los procesos de deceleración, reducir de marcha lo más tarde posible.
- Realizar siempre la conducción con anticipación y previsión.
- Recordar que mientras no se pisa el acelerador, manteniendo una marcha engranada, y una velocidad superior a los 20 km/h, el consumo de carburante es nulo.

Aplicando las reglas anteriores, se efectuará un menor número de cambios de marcha. En pruebas realizadas, se ha comprobado que circulando lo máximo posible en las marchas más largas se obtiene un ahorro comparativo del orden del 20% en el número de cambios realizados, lo que significa un ahorro en el uso del embrague, de los frenos, de la caja de cambios y del motor.

Se logra también con esta técnica un cambio de actitud en la conducción, creando un estilo de conducción menos agresivo, basado en la anticipación y en la previsión, que repercute en un menor grado de estrés para el conductor, y también, en una reducción del número de accidentes, como indican las cifras de los países europeos en los que está plenamente implantada la “conducción eficiente”.

Una recomendación importante a tener en cuenta por los conductores formados en las técnicas de la conducción eficiente consiste en llevar el control del consumo del vehículo a lo largo del tiempo. Este control se realizará mediante anotaciones de los

kilómetros recorridos y litros de carburante consumidos cada vez que se proceda a llenar el depósito.

Esta sencilla forma de actuar incrementa la eficacia de las técnicas de la conducción eficiente en el ahorro de carburante y logra conservar la actitud de ahorro evitando que se pierda con el transcurso del tiempo. Resulta, además, de mucha utilidad a la hora de realizar detecciones de averías al alertar sobre variaciones significativas de consumo de carburante.

2.3.4 El arranque e inicio de la marcha

[10] Para realizar el arranque de una forma correcta desde los puntos de vista tanto mecánico como de consumo, es conveniente arrancar el motor sin acelerar. Se gira la llave de contacto e inmediatamente la regulación del motor ajusta las condiciones necesarias para un arranque efectivo. En un automóvil moderno se realizan de forma automática todos los preparativos necesarios para el arranque del coche. Por tanto, acelerar cuando se arranca el motor sólo sirve para desajustar la regulación electrónica y restar rendimiento a la operación del arranque.

En los vehículos propulsados por motores de gasolina se ha de iniciar la marcha inmediatamente después de arrancar el motor. Esperar parado con el motor en marcha no aporta ninguna ventaja, ya que ralentiza el calentamiento del motor.

En los vehículos diesel conviene esperar unos segundos, una vez que se ha arrancado el motor, antes de comenzar la marcha. Con ello se logra que llegue el aceite en condiciones adecuadas a la zona de lubricación.

2.3.5 El tacómetro o cuentarrevoluciones

[10] Es el indicador clave a seguir cuando realizamos los cambios de marchas, así como para controlar el desarrollo de la conducción. En el tacómetro podemos observar las rpm del motor en cada instante.

2.3.6 Realización general de los cambios de marchas

[10] En los procesos de aceleración, cambiar de forma rápida hasta la marcha más larga en la que se pueda circular. El cambio de marchas se puede llevar a cabo teniendo en cuenta las revoluciones del motor o bien la velocidad a la que circulemos:

Según las revoluciones:

- En los motores de gasolina: entre 2.000 y 2.500 rpm.
- En los motores diesel: entre las 1.500 y 2.000 rpm.

Según la velocidad:

- 1ª marcha: solo para iniciar el desplazamiento.
- 2ª marcha: a los 2 segundos o 6 metros.
- 3ª marcha: a partir de unos 30 km/h.
- 4ª marcha: a partir de unos 40 km/h.
- 5ª marcha: a partir de unos 50 km/h.

En los procesos de deceleración, reducir una marcha lo más tarde posible levantando el pie del acelerador y efectuando las pequeñas correcciones necesarias con el pedal de freno.

2.3.7 Las marchas largas

[10] En la figura **2.5,** que se muestra a continuación, podemos apreciar la importancia de la utilización de las marchas largas en la conducción. La figura se compone de dos gráficos que muestran, a la velocidad constante de 60 km/h, el ahorro en carburante que supone el circular con marchas más largas, teniendo en cuenta la cilindrada del vehículo:

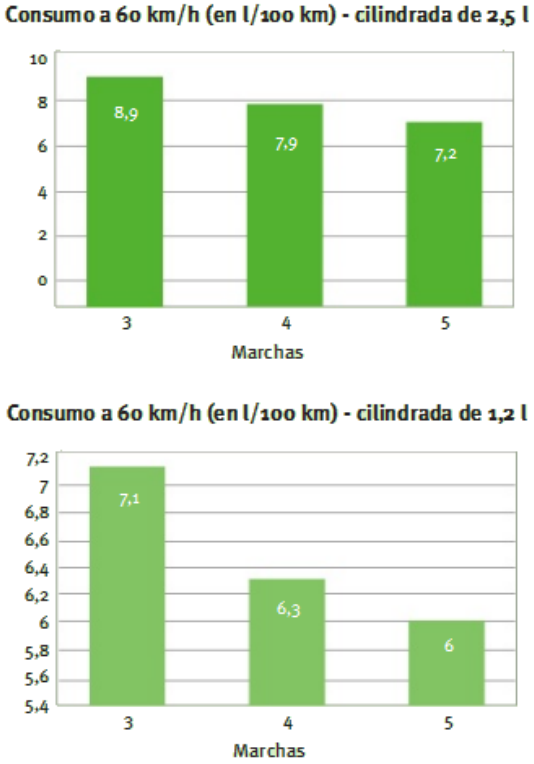


Figura 2.5. Ahorro carburante marchas largas [10]

Se puede observar en la **figura 2.5** que el ahorro que supone circular en 4º marcha en lugar de circular en 3ª, sobrepasa en ambos casos el 10%, mientras que si se circula en la 5ª marcha, supone un ahorro de carburante del 15% en la menor cilindrada y hasta un 20% en la mayor.

De estos datos podemos extraer dos conclusiones, la principal es que cuanto más larga es la marcha con la que se circula, siempre por encima de un número mínimo de revoluciones del motor, menor consumo de combustible. La segunda conclusión que deducimos consiste en que a mayor cilindrada, mayor impacto en el consumo tiene circular en una marcha más larga.

El número de revoluciones mínimo para circular en una marcha es de 1.000 rpm, menos en la 5ª marcha cuyo mínimo de revoluciones es de 1.500 rpm.

La relación de marchas indicada anteriormente no es válida si el vehículo circula por una vía con posibles incorporaciones a baja velocidad y tampoco si el vehículo está cargado en exceso.

Por otro lado, no se deben realizar frenadas innecesarias que conllevarán posteriormente sus correspondientes aceleraciones. Se debe mantener una velocidad de circulación lo más uniforme posible.

Se puede ver a continuación una gráfica representativa (**figura 2.6**) de los consumos (en litros/100 km) relacionados con las velocidades a las que se circula estando en una marcha determinada:

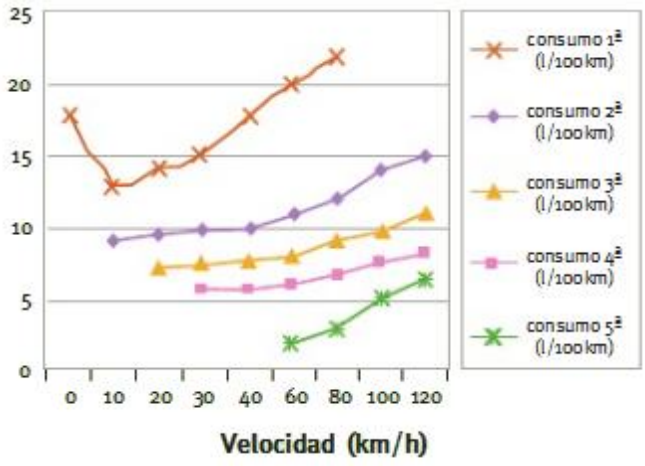


Figura 2.6. Relación marchas con el consumo [10]

Respecto a las altas velocidades, es importante notar que el consumo depende de la velocidad elevada al cuadrado. Aumentar la velocidad en un 20% (por ejemplo, pasar de 100 a 120 km/h) significa un aumento del 44% en el consumo. Pasaríamos de consumir unos 8 l/100 km a 11,5 l/100 km.

2.3.8 El freno de motor

[10] Cuando se tenga que decelerar ante una disminución de la velocidad de circulación de la vía, o ante una posible detención, se utilizará:

- El “freno motor”, si es posible, sin reducción de marcha.
- El freno de pié para realizar pequeñas correcciones puntuales necesarias para acomodar la velocidad o para la detención final.

El concepto de “freno motor” consiste en dejar el vehículo rodar por su propia inercia, con una marcha engranada y sin pisar el acelerador. Sólo cuando sea realmente necesario, se acompañará de una reducción de marcha. Así logramos que las ruedas, en vez de ser receptoras de par del motor, arrastren al motor en su movimiento de giro. La resistencia del motor a girar actúa entonces como freno, provocando una reducción progresiva de la velocidad del vehículo. A mayores revoluciones del motor, es decir, con marchas más cortas, mayor fuerza de retención y, por tanto, mayor reducción de velocidad. Siempre que sea posible, se utilizará el proceso de deceleración definido por los siguientes pasos:

- Levantar el pie del acelerador.
- Dejar el coche rodar por su propia inercia con la marcha engranada.
- Posicionar el pie sobre el pedal de freno y efectuar las pequeñas correcciones necesarias para acomodar la velocidad.

Siguiendo el proceso descrito se experimenta un frenado progresivo con un menor desgaste del embrague y de la caja de cambios y, sobre todo, un menor consumo de carburante. Al no reducir de marcha, se evita pasar por el punto muerto en el cual el consumo de combustible no es nulo (motor a ralentí). Aunque el consumo provocado sólo por un cambio de marcha no sea muy elevado, si se añaden los consumos de todas las reducciones de marcha inútiles en procesos de deceleración, se obtiene un consumo total de cierta relevancia.

2.3.9 Anticipación

[10] A través de la anticipación, junto con una adecuada distancia de seguridad, es posible reconocer las características del tráfico y sus potenciales situaciones, con lo que se tendrá más tiempo de reacción ante posibles imprevistos derivados del entorno considerado. La anticipación permite advertir a tiempo las situaciones peligrosas y adoptar oportunamente medidas para esquivar situaciones inminentes.

Una actitud anticipativa supone menor cansancio para el conductor, habitualmente sometido al estrés generado por las ciudades de tráfico denso y complejo, así como por la agresividad que pueden mostrar los conductores circundantes.

La anticipación se pone en práctica cuando:

- Se circula con un amplio campo de visión de la vía y de las circunstancias de la circulación. Un campo de visión adecuado es el que permite ver 2 o 3 coches por delante del propio.

- Se guarda una adecuada distancia de seguridad. En ciudad, a 50km/h, ha de ser mayor de 2 segundos o 30 metros de distancia. En carretera, a 100 km/h, será de 3 segundos u 80 metros de distancia.

2.3.10 Tramos con pendiente

[10] Las técnicas de la conducción eficiente enunciadas hasta ahora hacen referencia a una conducción desarrollada en terreno llano. Se ha de hacer una mención especial al caso de la conducción en tramos que presenten pendiente, ya sean de bajada o de subida.

a) Pendiente descendente:

Cuando en una vía con pendiente descendente se realiza un proceso de aceleración, se realizarán los cambios de marcha a un número más bajo de revoluciones que el indicado anteriormente, al venir ayudado el proceso de aceleración por la pendiente de la vía.

En las pendientes pronunciadas, el uso del freno motor resulta muy importante para conseguir circular de un modo económico y seguro. El proceso óptimo es el siguiente:

- Sin reducir marcha, levantar el pie del acelerador y dejar bajar el coche rodando por su propia inercia.
- Si se mantiene la velocidad controlada, continuar en la marcha seleccionada.
- Si no se mantiene la velocidad controlada y se acelera en exceso el coche, realizar pequeñas correcciones puntuales con el freno de pie.
- En caso de no controlar la velocidad con el proceso anterior, si esta sigue aumentando pese a las correcciones con el freno, se procederá a reducir a una marcha inferior.
- En la nueva marcha inferior, volver a repetir todos los pasos.

Una consideración a tener en cuenta es que nunca se ha de bajar una pendiente en punto muerto pues:

- Se incrementa el consumo de carburante, ya que el circular en ralentí supone un consumo de carburante, mientras que el freno motor no supone consumo alguno.
- Resulta peligroso, ya que obliga a solicitar de los frenos un mayor esfuerzo, suponiendo además un mayor desgaste de los mismos.

b) Pendiente ascendente:

En las vías de pendiente ascendente se ha de circular en la marcha más alta posible con el pedal del acelerador pisado hasta la posición que permita mantener la velocidad o aceleración deseada. Se reducirá a una marcha inferior, si el coche pierde velocidad, lo más tarde posible, pudiendo mantener la 5ª marcha hasta los 50 o 60 km/h.

Si se realiza un proceso de aceleración, se cambiará entonces de marcha a un número más alto de revoluciones, al venir frenado el proceso de aceleración por la pendiente que opone la vía.

2.3.11 Las 10 claves de la conducción eficiente

[10] A modo de resumen, a continuación, recogemos las recomendaciones del IDAE, estas son las 10 claves para una conducción eficiente:

1. **Arranque y puesta en marcha**: Arranque el motor sin pisar el acelerador. En los motores de gasolina iniciar la marcha inmediatamente después del arranque. En los motores diesel, esperar unos segundos antes de comenzar la marcha.

2. **Primera marcha**: Usarla sólo para el inicio de la marcha, cambiar a 2ª a los 2 segundos o 6 metros aproximadamente.

   Aceleración y cambios de marchas: Acelerar tras la realización del cambio.

   - **Según las revoluciones**: En los motores de gasolina: entre las 2.000 y 2.500 rpm. En los motores diesel: entre las 1.500 y 2.000 rpm.

   - **Según la velocidad**: a 2ª marcha: a los 2 segundos o 6 metros, a 3ª marcha: a partir de unos 30 km/h, a 4ª marcha: a partir de unos 40 km/h, a 5ª marcha: por encima de unos 50 km/h.

3. **Utilización de las marchas**: Circular lo máximo posible en las marchas más largas y a bajas revoluciones. Es preferible circular en marchas largas con el acelerador pisado en menor medida que en marchas cortas con el acelerador mas pisado. En ciudad, siempre que sea posible, utilizar la 4ª y 5ª marcha.

4. **Velocidad de circulación**: Mantener la velocidad lo más uniforme posible, buscar fluidez en la circulación, evitando frenazos, aceleraciones y cambios de marchas innecesarios. Por otro lado, también se debe moderar la velocidad ya que el consumo de carburante aumenta en función de la velocidad elevada al cuadrado.

5. **Deceleración**: Levantar el pie del acelerador y dejar rodar el vehículo con la marcha engranada en ese instante. Frenar de forma suave con el pedal del freno. Reducir de marcha lo más tarde posible, con especial atención en las pendientes descendentes.

6. **Detención**: Siempre que la velocidad y el espacio lo permitan, detener el coche sin reducir previamente de marcha.

7. **Paradas**: En paradas prolongadas (mayores de 60 segundos), es recomendable apagar el motor.

8. **Anticipación y previsión**: Conducir siempre con una adecuada distancia de seguridad y un amplio campo de visión que permita ver 2 o 3 vehículos por delante del nuestro. En el momento en que se detecte un obstáculo o una reducción de la velocidad de circulación en la vía, levantar el pie del acelerador para anticipar las siguientes maniobras.

9. **Seguridad**: En la mayoría de las situaciones, aplicar las reglas de la conducción eficiente contribuye al aumento de la seguridad vial. No obstante, existen circunstancias que requieren acciones específicas distintas, para que la seguridad no se vea afectada.

2.4 Aplicaciones de ahorro de combustible para Smartphones

El consumo de combustible de los vehículos a motor representa un gasto muy importante para los consumidores, puede representar al año algunos miles de Euros. La forma más común de ahorrar en este aspecto es buscar gasolineras más baratas o conducir a una velocidad moderada.

En este sentido, existen muchas aplicaciones que persiguen ayudar al usuario a reducir el gasto en combustible. Existen varios tipos de aplicaciones, desde aquellas que manejan bases de datos con las gasolineras existentes y el precio al que venden el combustible, pasando por aplicaciones que tratan de ofrecer rutas hacia el destino libres de atascos o que sean lo más cortas posibles, también existen aplicaciones que intentan que el usuario ahorre combustible circulando en la marcha adecuada para una determinada velocidad, o bien, que informan acerca de aceleraciones excesivas que aumentan el consumo.

En este apartado vamos a hacer un recorrido por algunas de estas aplicaciones disponibles para smartphones, independientemente de SO sobre el que corran.

2.4.1 WhatGas Precios Gasolina

Esta aplicación permite encontrar la gasolinera más cercana y con los mejores precios para cada uno de los distintos tipos de combustible. Se trata de una aplicación disponible para Android de manera gratuita.

Esta plataforma sirve para España, Portugal, Alemania y otros países. Los propios usuarios pueden actualizar la información acerca de los precios y demás datos de las gasolineras, esta información se guarda en la base de datos y los demás usuarios la obtendrán cuando realicen sus consultas. La búsqueda de la gasolinera la realiza el propio usuario explorando el mapa. En la **figura 2.7** podemos ver un par de capturas de la pantalla de esta aplicación.

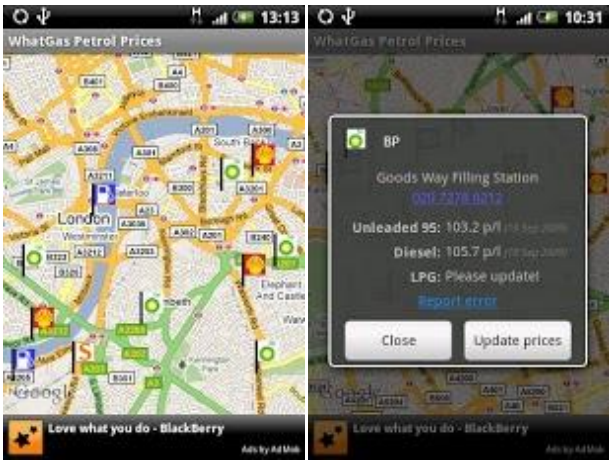


Figura 2.7. Capturas de pantalla de la aplicación WhatGas

2.4.2 Gasolineras España

Se trata de una aplicación para Iphone que ofrece información detallada y actualizada de las gasolineras más cercanas, es una aplicación gratuita. Estas son algunas características:

- Permite obtener la ruta más rápida para llegar a la gasolinera elegida.
- Almacena las gasolineras preferidas del usuario para un acceso rápido a la información de estas.
- Posee información sobre el horario de apertura, el precio del combustible actualizado o la situación en el mapa de cada una de las gasolineras.
- Realiza búsquedas con filtros de distancia hacia la gasolinera, por fecha de actualización de precios o por la marca de la gasolinera, entre otros.
- En la última versión se puede incluir el consumo medio de tu vehículo y en función de la distancia calcula el coste de viajar hasta esa gasolinera.

En la **figura 2.8** podemos observar algunas imágenes de esta aplicación, capturadas durante su ejecución.

Figura 2.8. Capturas pantalla aplicación Gasolineras España

2.4.3 Waze

Waze es una aplicación de navegación basada en una comunidad de 30 millones de usuarios. La información del tráfico se genera en tiempo real a través de los usuarios de la comunidad, de esta forma, puedes obtener la mejor ruta hacia el destino. Si conduces con Waze abierta aportarás datos en tiempo real sobre el tráfico y la vía. Algunas características de esta aplicación:

- Permite alertar de accidentes, obstáculos, policía y otros eventos.
- Permite consultar los precios de las gasolineras, actualizados por la comunidad.
- Realiza enrutamientos basándose en la información sobre el tráfico actual.
- Guía de navegación vocal.
- Si cambia el tráfico, se recalcula la ruta.
- Los mapas se actualizan por medio de la comunidad de editores de mapas.

Se trata de una aplicación gratuita, disponible en Google Play, desarrollada para Android. En la **figura 2.9** podemos ver algunas pantallas de esta aplicación.

Figura 2.9. Capturas de pantalla de Waze

2.4.4 EcoShifter

EcoShifter es una aplicación de ahorro de combustible en tiempo real, mientras conduces. Esta aplicación se conecta al vehículo utilizando un adaptador Bluetooth y de esta forma accede a los datos de este, velocidad, rpm, marcha engranada, etc. Sólo es apta para vehículos que dispongan de una centralita *On Board Diagnostics* (OBD2). Es una aplicación para Android que se puede descargar de manera gratuita. Algunas características de esta aplicación son:

- Indicador de cambio de marcha.
- Alerta de frenazo brusco y aceleración brusca.
- Barra de RPM que cambia de color.
- Indicador de consumo (L/100Km).
- Alertas por voz.

En la **figura 2.10** podemos ver algunas capturas de pantalla de esta aplicación, obtenidas durante su ejecución.

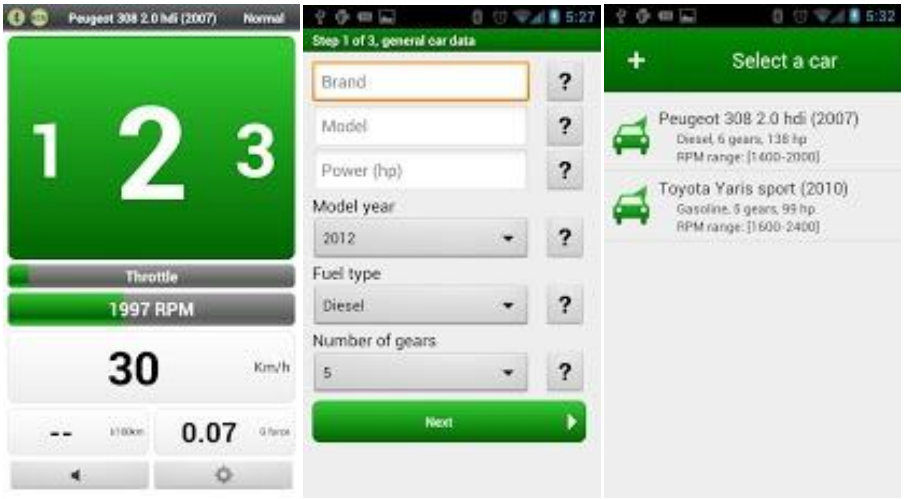


Figura 2.10. Capturas de pantalla de EcoShifter

2.4.5 DRIVEtools

Se trata de una aplicación que informa al usuario sobre la velocidad, rumbo y consumo, a la vez que alienta a conducir de una manera más suave, intentando evitar un gasto excesivo en el combustible. Dispone de una barra de conducción que indica si el modo de conducción es el adecuado para conseguir un consumo y unas emisiones reducidas.

Es una aplicación para Android gratuita. Aparentemente, uno de sus objetivos principales es indicar al conductor lo que ahorraría si utilizara un Toyota híbrido.

Algunas características adicionales son:

- Dispone de información sobre las gasolineras, de forma que puedes encontrar la más barata.
- Ofrece la posibilidad de almacenar la ubicación actual del vehículo para que sea más fácil encontrarlo cuando vuelves a utilizarlo (recuerda el lugar de estacionamiento).
- Puede almacenar datos acerca del vehículo, del seguro, etc.

En la **figura 2.11** se muestran algunas pantallas de ejemplo de esta aplicación.

Figura 2.11. Capturas de pantalla de DRIVEtools

2.4.6 DriveGain

Es una aplicación que aporta consejos para conducir de forma eficiente. Se trata de una aplicación para Iphone que cuesta alrededor de 5,5€ su descarga.

La aplicación empieza solicitando el modelo del vehículo, para obtener sus datos técnicos de una base de datos. Durante la ejecución, dicha plataforma envía al usuario mensajes de voz acerca de su manera de conducir.

Se basa en tres parámetros: la marcha insertada, la suavidad con el freno y la suavidad con el acelerador.

Según los desarrolladores, siguiendo sus consejos el usuario puede ahorra unos 175€ al año.

A continuación, mostramos algunas capturas de pantalla de dicha aplicación en la **figura 2.12**.

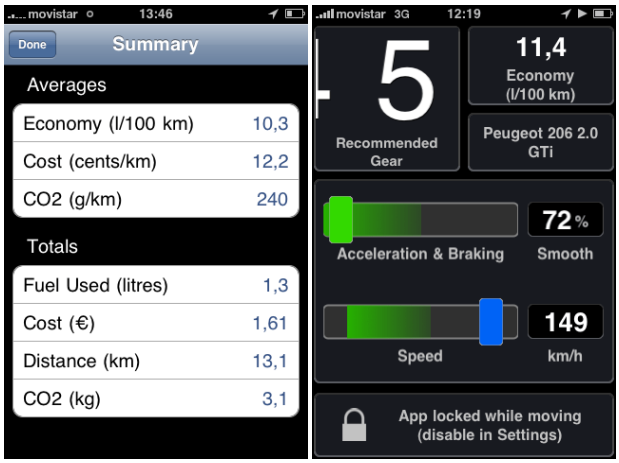


Figura 2.12. Pantallas de DriveGain

2.4.7 Nuestra aplicación

En el presente proyecto se ha desarrollado una aplicación con objeto de ayudar al usuario a realizar una conducción eficiente. Dicha aplicación sigue la línea de DriveGain y de EcoShifter, al igual que estas, nuestro sistema va informando al usuario en tiempo real, tanto a través de la pantalla del dispositivo como mediante avisos sonoros, acerca de si sus acciones son propias de una conducción eficiente o no.

Nuestra aplicación basa su funcionamiento en las 10 claves para una conducción eficiente según el IDAE. Esto afecta a la relación de marchas, a la suavidad de las aceleraciones y desaceleraciones, a la circulación por tramos con pendiente, etc.

Cabe destacar de la plataforma desarrollada que esta es completamente configurable por el usuario, de modo que es este el que establece donde se fija la línea entre el ahorro y el confort durante el viaje. El usuario puede configurar los límites que no quiere rebasar haciéndolos más restrictivos o menos.

La aplicación permite, incluso, adaptar esos límites al tipo de vía, no es lo mismo conducir por las calles de una ciudad que conducir por una carretera con límite de velocidad 70 Km/h o hacerlo en una autovía con límite de 120 Km/h. Evidentemente, siempre se ahorrará mas combustible viajando a 70 Km/h que a 120 Km/h, pero, en una vía que tenga fijado el segundo caso como límite de velocidad circular a 70 Km/h puede ser peligroso si el resto de vehículos circulan a una velocidad mayor ya que se pueden producir accidentes por alcance o similar.

Otro punto que cubre nuestra aplicación y que no lo hacen las mencionadas tiene que ver con los tramos en pendiente. Los tramos en pendiente afectan al consumo del vehículo de manera significativa, en una pendiente ascendente el consumo aumenta, circular en una cuesta arriba a la misma velocidad que lo haces en un llano supone un gasto extra de combustible, todavía se acentúa aún más esta situación durante una aceleración.

Por otra parte, en un tramo de pendiente descendente, como hemos visto en el apartado sobre conducción eficiente (**2.3**), si circulamos a una velocidad superior a 20 Km/h, teniendo una marcha engranada, y soltamos el pedal del acelerador nuestro consumo será nulo. De esta forma, puede darse el caso que circulemos a 100 Km/h con el motor moviéndose a un régimen de revoluciones alto y si no pisamos el acelerador el consumo del vehículo en ese caso sería nulo. Las dos aplicaciones citadas anteriormente tratarían esta situación como si se circulara por un llano y dicha velocidad y régimen del motor lo obtuviéramos a través de pisar el acelerador.

Respecto a EcoShifter, nuestra aplicación presenta el inconveniente de que no tiene información concreta del funcionamiento del coche como son, las rpm, la marcha engranada, etc. Sin en cambio, la forma en la que EcoShifter obtiene esta información reduce sensiblemente su cuota de mercado, sólo funciona con coches que cumplen unos requisitos y, además, el usuario necesita adquirir un adaptador Bluetooth y conectarlo a su vehículo.

La aplicación desarrollada en el presente proyecto obtiene sus datos a través del GPS, esta cualidad penaliza en la precisión de los cálculos, pero, a cambio, convierte nuestro sistema en compatible con cualquier tipo de vehículo a motor y no supone ningún gasto extra.

Por otro lado DriveGain, obtiene también sus datos a través de la localización del teléfono. Una ventaja importante que presenta nuestra aplicación frente a DriveGain es que está desarrollada para Android por lo que es compatible con un número mayor de dispositivos. Además, el alto precio que presenta esta aplicación es una clara desventaja de cara al público interesado en adquirirla.

Capítulo 3

Análisis, diseño e implementación

En este tercer capítulo de la presente memoria vamos a centrarnos en la aplicación desarrollada. Comenzaremos comentando las decisiones tomadas a priori sobre las características de la aplicación a desarrollar, veremos la forma que tendrá nuestra plataforma, estudiaremos los diferentes casos de uso y los requisitos del sistema resultante.

Una vez conozcamos las decisiones iniciales nos centraremos en el diseño y la implementación de la aplicación. Para ello, descompondremos nuestra plataforma en sus componentes fundamentales, describiendo en detalle cada uno de ellos.

3.1 Análisis

Durante este apartado conoceremos las decisiones tomadas a priori sobre la aplicación que será desarrollada. Comenzaremos con un breve resumen de los objetivos principales que queremos conseguir, después, incluiremos una descripción de la forma básica que presentará la plataforma que vamos a desarrollar. Posteriormente, veremos en detalle los distintos casos de uso de un posible usuario y también se detallarán los requisitos que deberá cumplir la aplicación. Estos requisitos serán evaluados en el capítulo 5, el cual se ha destinado a contener los resultados de las diferentes pruebas que serán necesarias para comprobar el correcto funcionamiento de la aplicación realizada.

3.1.1 Introducción

La aplicación a desarrollar en el presente proyecto utilizará el SO Android. El objetivo principal de la plataforma es ayudar al usuario a ahorrar combustible mientras conduce un vehículo a motor, impulsado por gasolina o gasoil.

Para lograr el objetivo principal, esta aplicación se basará en las claves de la conducción eficiente. La aplicación desarrollada deberá tener en cuenta el mayor número posible de técnicas sobre conducción eficiente, además, informará de estas al usuario mientras la aplicación este corriendo con objeto de influir en el comportamiento del conductor (nuestro usuario).

La aplicación desarrollada constará de tres pantallas principales. En la primera pantalla, llamada “pantalla de configuración”, el usuario podrá configurar las restricciones sobre la conducción que quiera aplicar. Se podrá configurar la aplicación de manera más restrictiva produciéndose así un ahorro significativo, o bien, se podrá establecer una configuración más permisiva que permitirá al usuario irse familiarizando con las técnicas de conducción eficiente. El usuario, con el tiempo, aprenderá realizar una configuración que aúne ahorrar combustible y el confort durante el viaje. La

aplicación establecerá una configuración "por defecto" como punto inicial recomendable sobre el que trabajar en la conducción eficiente.

En la segunda pantalla, nos referiremos a ella como "pantalla principal", se centrará la funcionalidad general de la aplicación. En esta pantalla se mostrarán las indicaciones sobre la conducción en tiempo real, se establecerán indicaciones visuales y sonoras. Dicha pantalla mostrará datos sobre la situación actual (aceleración, velocidad y pendiente) y con estos datos calculará la forma de actuar del conductor. Se expondrán indicaciones sobre la marcha que debería engranar en la caja de cambios, aceleración, deceleración, velocidad, pendiente, variación del consumo debido a la pendiente y tiempo que el vehículo se encuentra detenido.

Por último, la tercera pantalla, a la cual llamaremos "pantalla resumen", contendrá un resumen sobre el viaje realizado, mostrará los valores medios y los valores máximos relativos a la velocidad y la aceleración. También podremos encontrar en ella la distancia recorrida y la pendiente media durante el viaje. Además, ofrecerá al usuario algunas conclusiones sobre los datos obtenidos.

Por otro lado, todas las pantallas contarán con una ventana emergente en la que se detallará información sobre la pantalla en la que se encuentra el usuario, a modo de ayuda. En el caso de la tercera pantalla, en vez de mostrar instrucciones sobre el manejo de dicha interfaz, en realidad mostrará al usuario algunos consejos sobre conducción eficiente.

3.1.2 Casos de uso

Para empezar, un caso de uso se asocia a un actor, es el objetivo del actor usando el sistema lo que el caso de uso quiere describir. Es decir, un caso de uso es la forma en la que el actor (en este caso el usuario) utiliza el sistema. Se podría resumir como una lista de usos típicos que le da el usuario al sistema. En la **figura 3.1** podemos ver los diferentes casos de uso referentes a la aplicación desarrollada en este proyecto.

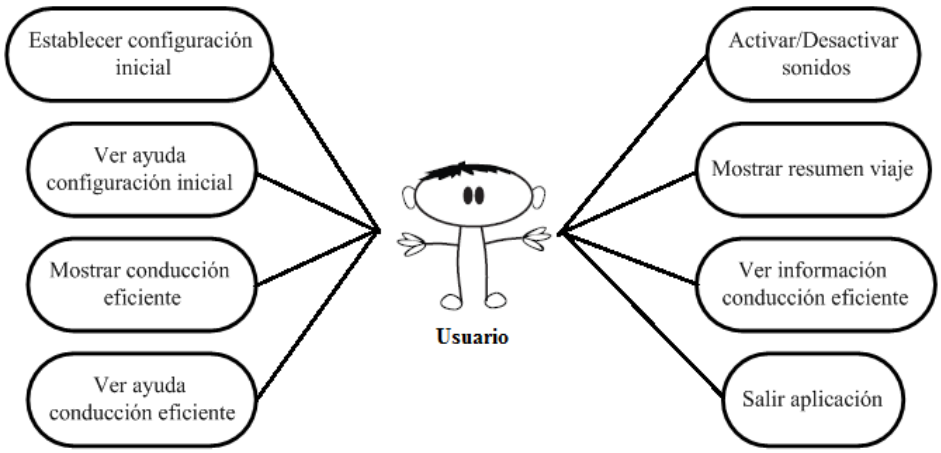


Figura 3.1. Casos de uso

A continuación, describiremos un poco más en profundidad cada uno de los casos de uso descritos en la **figura 3.1**. Para ello, estableceremos una serie de atributos pertenecientes a cada caso, los atributos serán:

- **Identificador**: Cada caso de uso tendrá un identificador único, en este caso, comenzará por "CU-" seguido de un número entero.

- **Nombre**: El nombre del caso de uso que se está describiendo.

- **Objetivo**: Una breve descripción acerca del objetivo que el usuario persigue en este caso de uso.

- **Pre-condiciones**: Condiciones previas que deben darse para que se pueda llevar a cabo este caso de uso.

- **Rumbo normal**: Descripción de los pasos a seguir para conseguir el objetivo.

- **Post-condiciones**: Estado posterior del sistema después de realizarse el caso de uso.

Cada uno de los casos de uso descritos en la **figura 3.1** estará contenido en una tabla independiente. Estos son los diferentes casos de uso (**tablas desde la 3.1 hasta la 3.8**):

Identificador	**CU-01**
Nombre	Establecer configuración inicial
Objetivo	Fijar los valores medios y máximos para los límites de aceleración y velocidad que se utilizarán durante la ejecución de la aplicación
Pre-condiciones	Haber arrancado la aplicación y encontrarse en la pantalla de configuración
Rumbo normal	1. Arrancar la aplicación, aparecerá como pantalla de inicio, o bien, 1. Presionar la opción "pantalla previa" en el menú de la pantalla principal 2. Modificar los valores mediante la manipulación de las barras de búsqueda de progreso específicas para cada valor
Post-condiciones	Cuando el usuario pulse el botón "comenzar" se lanzará una ventana de confirmación, en el caso de presionar "aceptar", en dicha ventana, se establecerá la configuración creada

Tabla 3.1. Caso de uso CU-01

Identificador	CU-02
Nombre	Ver ayuda configuración inicial
Objetivo	Mostrar una ventana emergente que contiene una breve descripción relativa a cada uno de los campos que hay que configurar y otros datos del funcionamiento de esta pantalla
Pre-condiciones	Encontrarse en la pantalla de configuración de la aplicación
Rumbo normal	Seleccionar en el menú la opción "ayuda"
Post-condiciones	Una vez haya terminado, pulsar "OK", la ventana se cerrará y la vista volverá a la pantalla de configuración

Tabla 3.2. Caso de uso CU-02

Identificador	CU-03
Nombre	Aprender conducción eficiente
Objetivo	Indicar al usuario, a través de la pantalla y mediante sonidos, como debe actuar sobre su estilo de conducción para ajustarse a los parámetros establecidos en la configuración inicial y a los consejos sobre conducción eficiente en tiempo real
Pre-condiciones	1. Haber establecido una configuración en la pantalla dispuesta para este objetivo
Rumbo normal	1. Pulsar "comenzar" en la pantalla de configuración 2. Pulsar "aceptar" en la ventana emergente que se muestra, o bien, 1. Pulsar "volver" en el menú de la pantalla de resumen
Post-condiciones	Nos encontraremos en la pantalla principal de la aplicación, dicha pantalla mostrará información de ayuda a la conducción en tiempo real

Tabla 3.3. Caso de uso CU-03

Identificador	CU-04
Nombre	Ver ayuda conducción eficiente
Objetivo	Muestra una ventana de ayuda acerca de la pantalla principal de la aplicación, la cual se corresponde con la pantalla de asistencia a la conducción en tiempo real
Pre-condiciones	Encontrarnos en la pantalla principal de la plataforma
Rumbo normal	Pulsar en la opción "Ayuda" del menú de esta pantalla
Post-condiciones	Se abrirá una ventana en la que el usuario puede encontrar información acerca de como la aplicación se comunica con él para ayudarle a conducir. Si el usuario pulsa "OK" la ventana se cerrará y se volverá a mostrar la pantalla principal

Tabla 3.4. Caso de uso CU-04

Identificador	CU-05
Nombre	Activar/Desactivar sonidos
Objetivo	Activar o desactivar los avisos sonoros que se producen en la pantalla principal de ayuda a la conducción
Pre-condiciones	Encontrarnos en la pantalla principal de la plataforma
Rumbo normal	Pulsar en la opción "Activar/Desactivar sonido" del menú de esta pantalla
Post-condiciones	Si el sonido estaba activado se mostrará una ventana emergente indicando que se ha desactivado el sonido y no se emitirán los avisos sonoros. Si el sonido estaba desactivado ocurrirá lo contrario, se activará y se indicará mediante dicha ventana emergente

Tabla 3.5. Caso de uso CU-05

Identificador	CU-06
Nombre	Mostrar resumen del viaje
Objetivo	Mostrar la pantalla de resumen que expone los datos acerca del viaje realizado, así como algunos consejos referentes a los resultados obtenidos
Pre-condiciones	1. Haber corrido la aplicación el tiempo suficiente para que se hayan obtenido datos del viaje 2. Encontrarse en la pantalla principal
Rumbo normal	Pulsar la opción "Mostrar resumen viaje" en el menú de la pantalla principal
Post-condiciones	Se mostrará una pantalla con los datos mas relevantes del viaje

Tabla 3.6. Caso de uso CU-06

Identificador	CU-07
Nombre	Ver información sobre conducción eficiente
Objetivo	Desplegar una ventana que contiene indicaciones acerca de cómo puede el usuario realizar una conducción eficiente actuando sobre la velocidad, aceleración, relación de marchas, etc.
Pre-condiciones	Encontrarse en la pantalla de resumen del viaje
Rumbo normal	Pulsar la opción "Información" en el menú de dicha pantalla
Post-condiciones	Se mostrará una ventana con dicha información, si el usuario pulsa "OK" se volverá a la pantalla de resumen de viaje

Tabla 3.7. Caso de uso CU-07

Identificador	CU-08
Nombre	Salir de la aplicación
Objetivo	Salir de la aplicación y volver a la pantalla principal de Android
Pre-condiciones	Encontrarse en la pantalla de configuración de la aplicación
Rumbo normal	Pulsar la opción "Salir" en el menú de la pantalla de configuración
Post-condiciones	Se mostrará una pantalla de confirmación, en el caso de pulsar "Aceptar" se saldrá de la aplicación volviendo a la pantalla principal de Android

Tabla 3.8. Caso de uso CU-08

3.1.3 Requisitos del sistema

En este apartado vamos a especificar los requisitos que nuestra aplicación debe cumplir. Para la especificación de los requisitos se ha utilizado la plantilla de especificación de requisitos "**Volere" [16]**. Esta plantilla provee secciones por cada tipo de requisitos apropiados para los actuales sistemas de software. Existen dos tipos fundamentales de requisitos, según dicha plantilla:

- **Funcionales**: Son los sujetos fundamentales o esenciales que construyen la médula del producto. Estos requisitos describen lo que el producto tiene que hacer o las acciones de procesamiento que debe tomar.

- **No funcionales**: Son las propiedades que las funciones deben poseer, tales como el desempeño y la capacidad de uso.

Dentro de los requisitos no funcionales podemos encontrar varios grupos distintos de requisitos, en nuestro caso, hemos definido requisitos de las siguientes categorías:

- Requisitos de **percepción**: Contiene los requisitos de apariencia y de estilo.

- Requisitos de **capacidad de uso** y **humanidad**: Abarca los requisitos de facilidad de uso, personalización, aprendizaje, comprensión y de accesibilidad.

- Requisitos de **desempeño**: Incluye los requisitos de velocidad y latencia, seguridad, precisión o exactitud, confiabilidad, tolerancia a fallos, etc.

- Requisitos **operacionales** y **ambientales**: Alberga aquellos requisitos referentes al ambiente físico esperado y ambiental, requisitos de interfaz con sistemas adyacentes, de producción y lanzamiento.

- Requisitos de **seguridad**: Se incluyen en esta categoría los requisitos de acceso, integridad, privacidad, auditoría e inmunidad.

Para exponer en esta memoria los distintos requisitos que debe cumplir nuestra aplicación se ha diseñado una plantilla que contiene los siguientes campos (extraídos de la plantilla "**Volere"**):

- **Identificador**: Una etiqueta que identifica unívocamente a este requisito. Se compondrá de un prefijo compuesto por letras y un sufijo numeral. El prefijo indicará el tipo de requisito de la siguiente manera (**tabla 3.9**):

RFUN	Requisito funcional
RAPA	Requisito de apariencia
RFAU	Requisito de facilidad de utilización
RDES	Requisito de desempeño
ROPA	Requisito operacional y ambiental
RSEG	Requisito de seguridad

Tabla 3.9. Prefijos identificador requisitos

- **Tipo de requisito**: Se identificará el tipo de requisito entre los descritos anteriormente.

- **Evento / Caso de uso**: El evento o el caso de uso con el que este requisito se relaciona.

- **Descripción**: En qué consiste exactamente el requisito.

- **Razón fundamental**: El objeto de la existencia de dicho requisito, el por qué.

- **Criterio de evaluación**: Cómo se puede comprobar que dicho requisito se cumple.

- **Relevancia**: La trascendencia de dicho requisito. Se han establecido tres grados: esencial, deseable u opcional.

A continuación, vamos a enumerar y describir cada uno de los requisitos establecidos para la aplicación desarrollada en este proyecto, cada uno de ellos posee su propia tabla (**tablas desde la 3.10 hasta la 3.37**):

Identificador	RFUN-01
Tipo requisito	Requisito funcional
Evento / Caso de uso	CU-01
Descripción	La aplicación debe poder ser configurada por el usuario
Razón fundamental	Un mismo usuario puede querer que prime el confort, o bien, el ahorro de combustible. La aplicación ha de ser adaptable a las necesidades del usuario en cada momento y a los distintos perfiles de usuarios que la usen
Criterio de evaluación	Comprobar que modificando los valores de configuración la aplicación cambia su comportamiento, en función de dichos valores.
Relevancia	Deseable

Tabla 3.10. Requisito RFUN-01

Identificador	RFUN-02
Tipo requisito	Requisito funcional
Evento / Caso de uso	CU-03
Descripción	Mostrar información del viaje en tiempo real
Razón fundamental	El usuario, de esta forma, puede comprobar que la aplicación responde a la situación real del viaje
Criterio de evaluación	Con el emulador de Android podemos generar un caso donde conozcamos todas las variables y así podemos comprobar que la aplicación muestra realmente esta situación en su interfaz
Relevancia	Deseable

Tabla 3.11. Requisito RFUN-02

Identificador	RFUN-03
Tipo requisito	Requisito funcional
Evento / Caso de uso	CU-03
Descripción	Enseñar al usuario las claves de la conducción eficiente en tiempo real
Razón fundamental	El objetivo de la aplicación es que el usuario pueda aplicar estas claves de la conducción eficiente mientras conduce su vehículo
Criterio de evaluación	Durante la ejecución se puede comprobar que la aplicación ofrece información sobre conducción eficiente
Relevancia	Esencial

Tabla 3.12. Requisito RFUN-03

Identificador	RFUN-04
Tipo requisito	Requisito funcional
Evento / Caso de uso	CU-03
Descripción	Los márgenes se deben adaptar a la pendiente de la carretera
Razón fundamental	El consumo de combustible varía si circulamos en llano, por una pendiente ascendente o una descendente. Los márgenes de la conducción eficiente deben tener eso en cuenta
Criterio de evaluación	En el emulador podemos simular que el usuario circula por una pendiente ascendente o descendente. Podremos comprobar entonces si los límites varían en esas circunstancias
Relevancia	Deseable

Tabla 3.13. Requisito RFUN-04

Identificador	RFUN-05
Tipo requisito	Requisito funcional
Evento / Caso de uso	CU-06
Descripción	Se mostrará un resumen del viaje con los datos mas relevantes
Razón fundamental	El usuario puede aprender como mejorar su conducción para hacerla mas eficiente observando este resumen
Criterio de evaluación	Debe existir una pantalla resumen en la que encontraremos esos datos
Relevancia	Esencial

Tabla 3.14. Requisito RFUN-05

Identificador	RAPA-01
Tipo requisito	Requisito de apariencia
Evento / Caso de uso	Todos
Descripción	La vista se debe adaptar a la posición del dispositivo, ya sea horizontal o vertical
Razón fundamental	Para el usuario puede ser mas cómodo utilizar el dispositivo en posición vertical u horizontal, la aplicación debe ser legible en ambos casos con la misma facilidad
Criterio de evaluación	Cambiando el terminal de posición, la vista en la pantalla cambiará automáticamente y se mostrará la adecuada en cada caso
Relevancia	Deseable

Tabla 3.15. Requisito RAPA-01

Identificador	RAPA-02
Tipo requisito	Requisito de apariencia
Evento / Caso de uso	Todos
Descripción	Será compatible con diferentes tamaños de pantalla y diferentes densidades de estas
Razón fundamental	La aplicación debe mostrarse adecuadamente en los diferentes dispositivos que utilicen el sistema Android y que poseen pantallas de diferentes características
Criterio de evaluación	Comprobar la apariencia de la aplicación en pantallas diferentes en tamaño y densidad
Relevancia	Esencial

Tabla 3.16. Requisito RAPA-02

Identificador	RFAU-01
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	Todos
Descripción	La interfaz debe ser sencilla e intuitiva
Razón fundamental	La aplicación está destinada a cualquier tipo de persona, por lo tanto, su interfaz debe ser fácilmente comprensible y sencilla de manejar
Criterio de evaluación	Un grupo de personas heterogéneo puede probar la aplicación
Relevancia	Esencial

Tabla 3.17. RFAU-01

Identificador	RFAU-02
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	Todos
Descripción	El uso de la aplicación no debe requerir conocimientos previos
Razón fundamental	Cualquier persona debe poder utilizar la aplicación fácilmente sin que sea condición necesaria tener conocimientos de física o mecánica
Criterio de evaluación	Un grupo heterogéneo de personas puede probar la aplicación
Relevancia	Esencial

Tabla 3.18. Requisito RFAU-02

Identificador	RFAU-03
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	CU-03
Descripción	La aplicación no requerirá un gran nivel de atención del usuario
Razón fundamental	Debido a que se ha diseñado para interactuar con el usuario mientras conduce, es necesario que no demande la atención del usuario, el cual debe estar concentrado en la conducción
Criterio de evaluación	Un grupo heterogéneo de personas puede probar la aplicación
Relevancia	Esencial

Tabla 3.19. Requisito RFAU-03

Identificador	RFAU-04
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	Todos
Descripción	El ahorro de combustible producido provocará que el usuario quiera seguir utilizando la aplicación
Razón fundamental	Comprobar que las técnicas aplicadas realmente producen un ahorro significativo hará incrementar la popularidad de la plataforma
Criterio de evaluación	Se puede comprobar si hay diferencia en el consumo del vehículo realizando el mismo trayecto dos veces, una de ellas siguiendo las indicaciones de la plataforma y la otra sin hacerlo
Relevancia	Esencial

Tabla 3.20. Requisito RFAU-04

Identificador	RFAU-05
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	Todos
Descripción	La apariencia será sencilla para no incrementar el consumo de batería
Razón fundamental	La plataforma hace uso del sistema GPS por lo que consume rápido la batería, para compensar esta cualidad la interfaz de usuario ha de ser sencilla de forma que esta no requiera demasiados recursos
Criterio de evaluación	Observar que la interfaz no posee detalles gráficos complejos
Relevancia	Deseable

Tabla 3.21. Requisito RFAU-05

Identificador	RFAU-06
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	Todos
Descripción	La interfaz será auto-explicativa
Razón fundamental	El usuario podrá utilizar la plataforma siguiendo las instrucciones incluidas en la propia plataforma, no será necesario un manual de instrucciones externo ya que este se encuentra integrado en el sistema
Criterio de evaluación	Comprobar que todos los campos del interfaz se encuentran explicados en las pantallas de ayuda correspondientes
Relevancia	Deseable

Tabla 3.22. Requisito RFAU-06

Identificador	RFAU-07
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	CU-03
Descripción	Mensajes cortos, precisos y fácilmente comprensibles
Razón fundamental	Los mensajes que lanzará la aplicación cumplirán las condiciones descritas, así, el usuario puede tener una conciencia situacional completa rápidamente y sin esfuerzo
Criterio de evaluación	Comprobar que los mensajes muestran la información necesaria de una manera breve, además, deberán poseer un lenguaje sencillo
Relevancia	Deseable

Tabla 3.23. Requisito RFAU-07

Identificador	RFAU-08
Tipo requisito	Requisito de facilidad de utilización
Evento / Caso de uso	CU-03
Descripción	Mensajes de audio suficientemente espaciados en el tiempo y en la pantalla sólo mostrar la información necesaria
Razón fundamental	No se debe saturar al usuario de información, sólo se le deberá presentar la necesaria, además los mensajes de audio estarán separados en el tiempo para permitir al usuario atender al mensaje y adaptar la situación a esa información
Criterio de evaluación	Comprobar que los mensajes de audio no se solapan entre sí, por otro lado, la pantalla no debe mostrar información que no se pueda utilizar para ahorrar combustible
Relevancia	Deseable

Tabla 3.24. Requisito RFAU-08

Identificador	RDES-01
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	Procesará coordenadas recibidas con una frecuencia máxima de 1 coordenada/segundo
Razón fundamental	Cuantas más coordenadas procese la aplicación mayor precisión lograremos en los cálculos de la situación actual
Criterio de evaluación	Con el emulador de Android podemos reproducir una situación en la que llegan coordenadas al terminal con la frecuencia descrita y la aplicación será capaz de procesar todas las coordenadas introducidas
Relevancia	Deseable

Tabla 3.25. Requisito RDES-01

Identificador	RDES-02
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	La latencia ha de ser la mínima posible. Lo deseable es que se encuentre entre 1 y 2 segundos
Razón fundamental	Ayudar al conductor en tiempo real implica que la aplicación responda a una situación actual y no a una situación antigua que puede haber cesado ya, lo que provocaría desconcierto en el usuario
Criterio de evaluación	Durante el uso se puede comprobar fácilmente si la situación descrita por la aplicación se corresponde con la realidad en ese instante
Relevancia	Esencial

Tabla 3.26. Requisito RDES-02

Identificador	RDES-03
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	Indicaciones sonoras de la situación
Razón fundamental	Por razones de seguridad, el usuario no debe apartar los ojos de la carretera mientras conduce para mirar la interfaz de la aplicación, por eso se proporcionarán avisos sonoros de la situación actual
Criterio de evaluación	Comprobar que los avisos sonoros se corresponden con los avisos en pantalla
Relevancia	Esencial

Tabla 3.27. Requisito RDES-03

Identificador	RDES-04
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	Error tolerable en la localización del dispositivo menor de 25 metros
Razón fundamental	Errores mayores a 25 metros provocan resultados en los cálculos demasiado alejados de la realidad y por tanto serán intolerables para este tipo de aplicación
Criterio de evaluación	Comprobar que los datos mostrados en la pantalla son cercanos a los que se pueden observar en el propio vehículo
Relevancia	Esencial

Tabla 3.28. Requisito RDES-04

Identificador	RDES-05
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	Utilizar varias muestras para calcular la altura respecto al nivel del mar
Razón fundamental	La precisión vertical del GPS es notablemente peor que la horizontal, habrá que utilizar un algoritmo para minimizar el error vertical
Criterio de evaluación	Se puede comprobar el pseudocódigo de dicho algoritmo. En tiempo real, se puede comprobar que la pendiente calculada por la aplicación se corresponde con la real de la vía transitada
Relevancia	Deseable

Tabla 3.29. Requisito RDES-05

Identificador	RDES-06
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	La aplicación estará preparada para recibir coordenadas de manera no uniforme respecto al tiempo
Razón fundamental	Podemos establecer el tiempo mínimo deseado entre dos coordenadas recibidas, por el contrario, no podemos establecer el tiempo máximo entre la recepción de coordenadas consecutivas
Criterio de evaluación	A través del emulador de Android podemos comprobar que la aplicación se comporta bien aunque las coordenadas lleguen en instantes de tiempo no equidistantes entre sí
Relevancia	Esencial

Tabla 3.30. Requisito RDES-06

Identificador	RDES-07
Tipo requisito	Requisito de desempeño
Evento / Caso de uso	CU-03
Descripción	Descartar coordenadas antiguas
Razón fundamental	Puede suceder que al dispositivo lleguen coordenadas anteriores a la última recibida, esas coordenadas serán descartadas ya que se corresponden a una posición anterior a la actual
Criterio de evaluación	Se puede comprobar el algoritmo utilizado para descartar dichas coordenadas o comprobar el código fuente
Relevancia	Esencial

Tabla 3.31. Requisito RDES-07

Identificador	ROPA-01
Tipo requisito	Requisito operacional y ambiental
Evento / Caso de uso	CU-03
Descripción	El producto se utilizará dentro de un vehículo, este ha de encontrarse al aire libre fuera de una zona de sombra producida por un edificio
Razón fundamental	El sistema GPS necesita de visibilidad directa con los satélites, utilizarlo dentro de un edificio penaliza su precisión
Criterio de evaluación	Comprobaremos que el sistema funciona dentro del vehículo cuando este se encuentra fuera de cualquier tipo de edificio
Relevancia	Esencial

Tabla 3.32. Requisito ROPA-01

Identificador	ROPA-02
Tipo requisito	Requisito operacional y ambiental
Evento / Caso de uso	Todos
Descripción	La aplicación se diseñará para usarse en vehículos turismos
Razón fundamental	Las condiciones de conducción de estos vehículos son diferentes a las del resto, por ejemplo, de una motocicleta o un camión, por tanto, no puede asegurarse la efectividad de la plataforma en esos casos
Criterio de evaluación	No procede
Relevancia	Esencial

Tabla 3.33. Requisito ROPA-02

Identificador	ROPA-03
Tipo requisito	Requisito operacional y ambiental
Evento / Caso de uso	Todos
Descripción	Debe ser compatible con el mayor número posible de versiones de Android disponibles hasta la fecha y posteriores
Razón fundamental	El objetivo de la plataforma es divulgativo, por tanto, es importante no limitar el número de usuarios potenciales
Criterio de evaluación	Comprobar que la aplicación funciona en las diferentes versiones (APIs)
Relevancia	Opcional

Tabla 3.34. Requisito ROPA-03

Identificador	ROPA-04
Tipo requisito	Requisito operacional y ambiental
Evento / Caso de uso	Todos
Descripción	Permitirá atender llamadas o mensajes de cualquier tipo entrantes al dispositivo durante la ejecución de la aplicación, esta retomará posteriormente su funcionamiento normal
Razón fundamental	Es importante que el dispositivo no pierda su propósito principal (la comunicación), es apropiado no limitar esta
Criterio de evaluación	Se puede comprobar atendiendo una llamada mientras la aplicación corre en el dispositivo
Relevancia	Esencial

Tabla 3. 35. Requisito ROPA-04

Identificador	RSEG-01
Tipo requisito	Requisito de seguridad
Evento / Caso de uso	Todos
Descripción	No accederá a información sensible del usuario
Razón fundamental	El sistema no necesita información referente al usuario (lista de contactos, mensajes, etc.) para su funcionamiento
Criterio de evaluación	Al instalar la aplicación no se solicitan al usuario permisos para acceder a dicha información
Relevancia	Opcional

Tabla 3.36. Requisito RSEG-01

Identificador	RSEG-02
Tipo requisito	Requisito de seguridad
Evento / Caso de uso	CU-01
Descripción	Controlar la introducción de datos inconsistentes en la pantalla de configuración por parte del usuario
Razón fundamental	Se deben controlar los datos que introduce el usuario a la aplicación ya que podrían provocar errores y excepciones en tiempo de ejecución si son inconsistentes o tienen un formato inadecuado
Criterio de evaluación	En la pantalla de configuración intentar introducir datos erróneos
Relevancia	Esencial

Tabla 3.37. Requisito RSEG-02

3.2 Diseño e implementación de la aplicación

En este apartado mostraremos los detalles del desarrollo de la aplicación "***AhorroCombustible***", objeto del presente proyecto. Describiremos su arquitectura interna y, también, los algoritmos desarrollados más importantes. Avanzaremos en la descripción empezando a alto nivel y continuaremos hasta llegar a los detalles.

3.2.1 Arquitectura de la aplicación

Vamos a describir, a continuación, la arquitectura de la aplicación. Para ello, empezaremos con un diagrama que muestre los módulos de los que consta el sistema y la manera en la que interactúan entre ellos (**figura 3.2**).

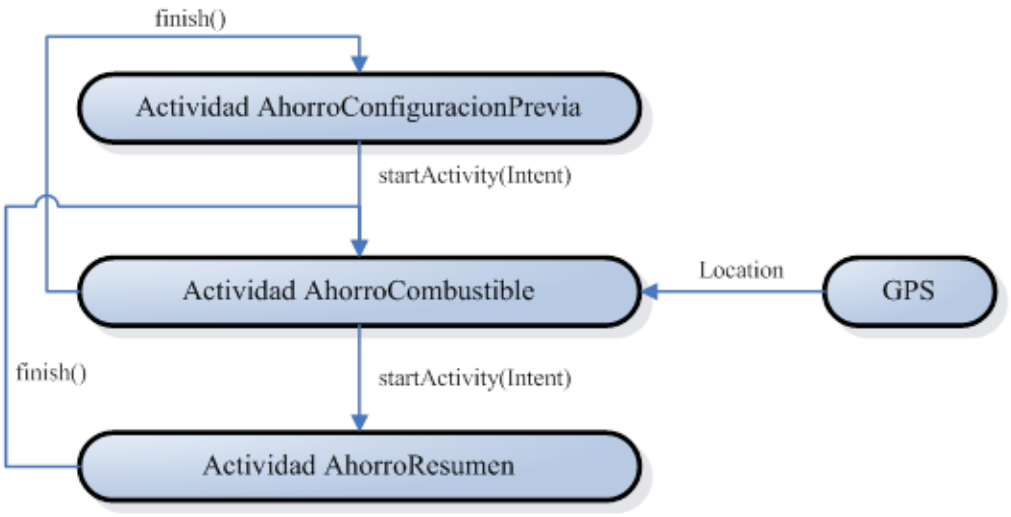


Figura 3.2. Arquitectura de la aplicación

Como se puede observar en la **figura 3.2**, la aplicación consta de tres actividades. La actividad principal recibe datos de un componente exterior, en este caso, el dispositivo GPS del aparato.

La actividad **AhorroConfiguracionPrevia** es la actividad que se lanza cuando arrancamos la aplicación. En la pantalla generada por esta actividad el usuario puede configurar algunos parámetros de la aplicación, una vez establecida la configuración, esta actividad llamará a la siguiente mediante el método ***startActivity(Intent)***, que contiene como argumento un objeto ***Intent*** (del API de Android ***android.content***), el cual, almacena información relevante para la siguiente actividad. De esta forma se crea la siguiente actividad.

La actividad creada se llama **AhorroCombustible**, se trata de la actividad que concentra la funcionalidad principal de la aplicación, debido a esto, nos hemos referido a ella a lo largo de esta memoria, como "pantalla principal" o "actividad principal". Esta actividad requiere para su funcionamiento datos relativos a la posición del dispositivo de modo que dicha actividad va a solicitarlos al sistema **GPS** instalado en el terminal. Estos datos serán recibidos en forma de objetos ***Location*** (del API de Android ***android.location***), los cuales, contendrán información relativa a la posición geográfica del dispositivo.

Por otra parte, encontramos la forma de interactuar de esta actividad con el resto de actividades. A través de esta actividad podemos volver a la pantalla anterior, generada por la actividad inicial, a través del método ***finish*()**, que destruirá la actividad actual. Puede también crear la actividad que genera el resumen llamando al método ***startActivity(Intent)***, como en el caso anterior este método contiene como argumento un objeto ***Intent***, aunque los datos que almacena dicho objeto son diferentes y serán descritos en apartados posteriores.

La actividad **AhorroResumen** se crea a través de la llamada al método ***startActivity(Intent)*** desde la actividad principal. La pantalla resultante es simplemente informativa y se genera a partir de los datos recibidos, contenidos dentro del objeto ***Intent***. Para terminar dicha actividad se llama al método ***finish*()**, el cual, destruye la actividad y devuelve la vista principal a la actividad anterior, que en este caso es la actividad **AhorroCombustible**.

3.2.2 Modelo de conocimiento de la aplicación

En este apartado mostraremos un diagrama de clases del sistema y describiremos un poco cada una de ellas, sólo se detallarán, en esta ocasión, las clases principales y los atributos más importantes. Por motivos de espacio se ha fragmentado el diagrama de clases completo en tres partes.

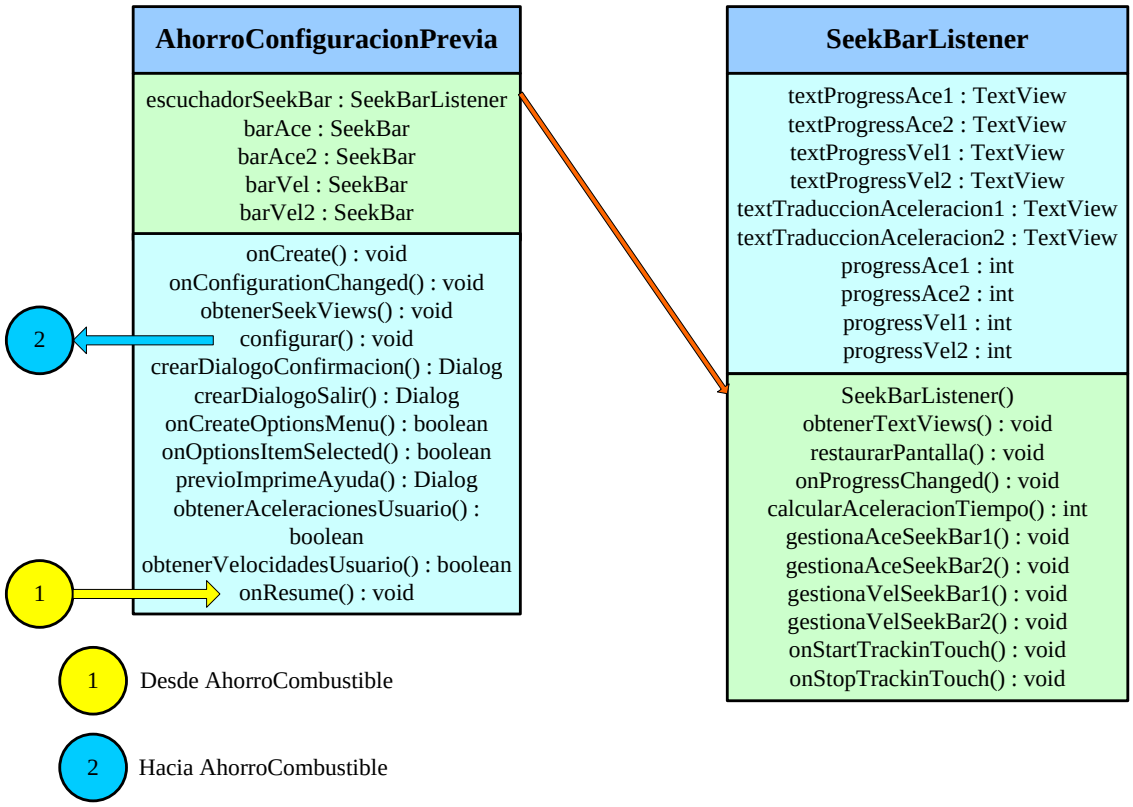


Figura 3.3. Diagrama de clases, 1/3

En la **figura 3.3** se muestran las dos primeras clases de un total de cinco que componen nuestra aplicación.

En primer lugar, tenemos la clase pública ***AhorroConfiguracionPrevia*** que hereda de la clase ***Activity*** del API de Android. Como se ha comentado en apartados anteriores, esta clase se encarga de mostrar una pantalla que le servirá al usuario para configurar algunos parámetros de la aplicación.

Esta primera pantalla se compone de algunas etiquetas y, además, como elementos principales contiene 4 objetos tipo ***SeekBar***, dos de ellos para configurar los límites de la velocidad y los otros dos para los límites de la aceleración. Los atributos ***barAce*** y ***barVel*** se utilizan para los límites recomendables, los atributos ***barAce2*** y ***barVel2*** sirven para configurar los límites máximos de estos dos parámetros.

El atributo ***escuchadorSeekBar*** es un objeto de la clase anidada a ***AhorroConfiguracionPrevia*** llamada ***SeekBarListener***. Este atributo nos va a permitir gestionar los cuatro objetos descritos en el párrafo anterior. Como función principal de esta clase, será la encargada de atender los eventos que surjan cuando el usuario actúe sobre uno de estos objetos.

Como se muestra en la **figura 3.3**, a través del método ***configurar()*** llamamos a la siguiente actividad, que en este caso es ***AhorroCombustible***. Esto se indica a través de una flecha saliente y un círculo azul. En el caso opuesto, cuando se vuelve de la

actividad principal, se invoca al método ***onResume()*** de esta clase, indicado en la **figura 3.3** con una flecha entrante y un círculo amarillo.

En la parte derecha de la **figura 3.3** encontramos la clase ***SeekBarListener***, esta clase implementa la interfaz ***OnSeekBarChangeListener*** del paquete ***android.widget.SeekBar***. Como se ha comentado anteriormente, esta clase se encarga de recoger los eventos lanzados por los objetos ***SeekBar*** de la clase ***AhorroConfiguracionPrevia***, también se encarga de gestionar las etiquetas que forman parte de la vista de esta actividad y que dependen del estado de dichos objetos. Se trata de una clase privada e interna de ***AhorroConfiguracionPrevia***.

Como atributos, la clase ***SeekBarListener*** posee algunas etiquetas que se muestran en la vista generada por esta actividad y una serie de enteros que almacenan el progreso seleccionado en cada uno de los objetos ***SeekBar***.

De los atributos mencionados en el párrafo anterior cabe destacar los llamados ***textTraduccionAceleracion1*** y ***textTraduccionAceleracion2***, estos se utilizan para que el usuario tenga una idea más clara de lo que implica el valor que está seleccionando, la aceleración se mide en m/s^2 pero esta medida no es intuitiva para un usuario inexperto, en estas etiquetas se muestra al usuario el tiempo que tardaría el vehículo, si la aceleración fuese constante, en alcanzar los 100 km/h partiendo desde el reposo. Esta forma de representar la aceleración es mucho más común al referirnos a un vehículo, más concretamente, al referirnos a la potencia proporcionada por el motor que impulsa este.

Siguiendo con la descripción de la clase ***SeekBarListener***, los atributos de tipo ***int*** (números enteros) que contiene esta clase almacenan el valor de progreso de cada barra. Las barras de progreso tienen un valor mínimo y un valor máximo, ambos de tipo ***int***, el valor del progreso es simplemente un número entero que se encuentra entre el máximo y el mínimo, ambos incluidos.

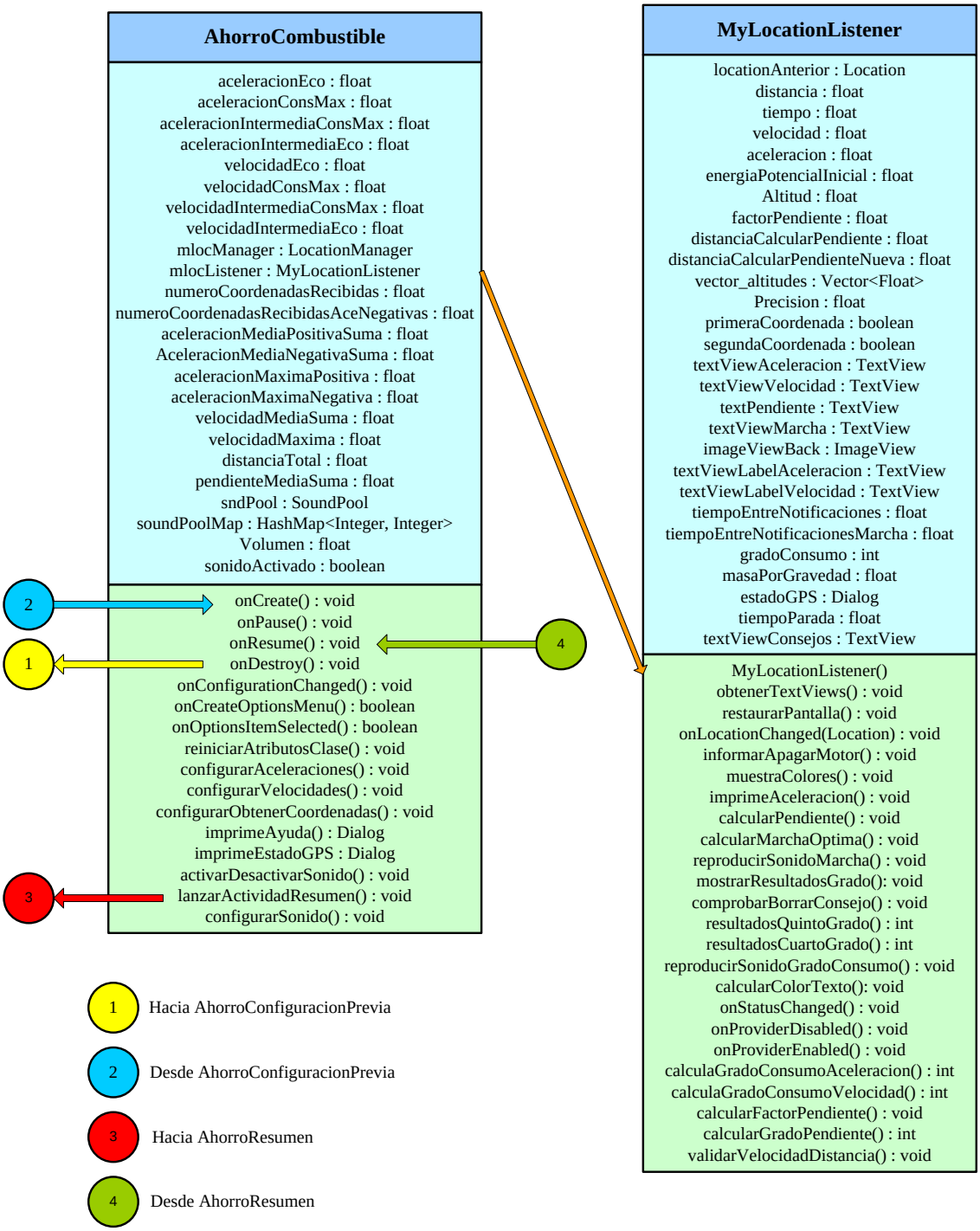


Figura 3.4. Diagrama de clases 2/3

En la **figura 3.4** se muestra la segunda parte del diagrama de clases que compone la aplicación desarrollada. Por un lado se muestra la clase pública ***AhorroCombustible***, esta concentra la funcionalidad principal de la aplicación tal como se ha comentado anteriormente. Por otro lado, se expone también la subclase ***MyLocationListener*** que se encuentra definida dentro de la clase ***AhorroCombustible***.

La clase ***AhorroCombustible*** hereda de la clase ***Activity*** de Android. Se encarga de gestionar la pantalla principal de la aplicación, en ella se muestra la situación actual del

viaje, además de consejos y avisos para el usuario, el fin de estos es ayudar al usuario a ceñirse a los límites establecidos en la pantalla de configuración y a las claves de la conducción eficiente.

Esta clase posee una serie de atributos de tipo ***float*** (coma flotante) que simplemente ayudan a realizar todos los cálculos sobre posición, velocidad, aceleración, pendiente, distancia, etc. Además, algunos de ellos tienen como objetivo servir para completar la pantalla de resumen generada por la siguiente actividad.

Merecen una atención especial los atributos llamados ***mlocManager*** y ***mlocListener***. El primero de ellos es un objeto de tipo ***LocationManager***, perteneciente al API ***android.location***, el segundo es de tipo ***MyLocationListener***, es un objeto de la clase interna de ***AhorroCombustible*** que implementa la interfaz ***LocationListener***, como veremos posteriormente. A través del primero podemos configurar como queremos recibir los objetos que contienen la localización, en nuestro caso, se ha configurado para obtener la localización a través del dispositivo GPS instalado en el terminal. El segundo atributo es el escuchador de eventos de localización, en nuestro caso, nos servirá para recibir los objetos ***Location*** enviados por el GPS a medida que se genera cada uno de ellos y según la configuración establecida mediante el objeto ***LocationManager***.

Los últimos cuatro atributos que podemos observar en la **figura 3.4**, dentro de la clase ***AhorroCombustible***, nos sirven para gestionar todo lo referente al audio de la aplicación, ya sea, utilizar sonidos, activar o desactivar el audio, gestionar el volumen de este, etc.

El atributo ***sndPool*** es un objeto tipo ***SoundPool*** del API ***android.media***. Este objeto nos permite cargar en memoria sonidos cortos, se cargarán en memoria sin codificar con un formato sencillo para que se puedan reproducir con una latencia baja y consumiendo pocos recursos. Además, este objeto permite gestionar si queremos reproducir varios sonidos a la vez, priorizar unos sonidos frente a otros, etc.

Mediante el ***HashMap*** llamado ***soundPoolMap***, podemos acceder rápidamente a estos sonidos almacenados en la memoria a través de su identificador, compuesto en este caso por un número entero. A través del atributo ***sndPool*** reproduciremos el sonido obtenido del ***HashMap***.

Como se ha descrito en un párrafo anterior, el atributo ***mlocListener*** es un objeto de tipo ***MyLocationListener***. ***MyLocationListener*** es una clase anidada a la clase ***AhorroCombustible***. Esta clase implementa la interfaz ***LocationListener***, tiene como cometido recibir notificaciones desde el objeto ***LocationManager***, cada vez que este genere un evento.

La clase ***MyLocationListener*** se encarga de recibir y procesar los datos de localización, gestiona todo lo referente a los datos recibidos del GPS, hace los cálculos necesarios con esos datos y a partir de los resultados obtenidos actualiza la vista

generada por la actividad ***AhorroCombustible***, también se encarga de reproducir los sonidos y mostrar los mensajes pertinentes en cada circunstancia.

Los atributos de esta clase son los objetos de tipo ***TextView***, que forman parte de la vista de esta actividad y, también, una serie de atributos numerales, estos sirven para realizar todos los cálculos necesarios para procesar la situación actual recibida a través del GPS. Algunos de estos atributos serán descritos en próximos apartados ya que se emplean en los algoritmos más importantes que se han elaborado para implementar esta aplicación.

A parte de dichos atributos, hay un atributo llamado ***locationAnterior***, es un objeto de la clase ***Location*** que almacena el último objeto de esta clase recibido y procesado, este atributo nos sirve para comparar algunos datos suyos con los que contiene el nuevo objeto de este tipo que acabamos de recibir, este proceso también será descrito en el siguiente apartado.

Por último, de la **figura 3.4** podemos destacar las relaciones entre clases. De nuevo, estas relaciones se muestran con flechas entrantes y salientes junto con círculos que contienen un número. Los colores amarillo y azul representan la forma en la que ***AhorroCombustible*** se relaciona con ***AhorroConfiguraciónPrevia***. Cuando abandonemos esta actividad volveremos a la actividad anterior (flecha amarilla), debido a la pila de actividades de Android.

Por otra parte, la flecha roja y la flecha verde representan la relación de la clase ***AhorroCombustible*** con la clase ***AhorroResumen***. En este caso, el método ***lanzarActividadResumen*()** es el encargado de lanzar la siguiente actividad (flecha roja). Cuando se abandone dicha actividad se volverá de nuevo a ***AhorroCombustible*** mediante el método ***onResume*()** (flecha verde) mostrando esta actividad en la pantalla.

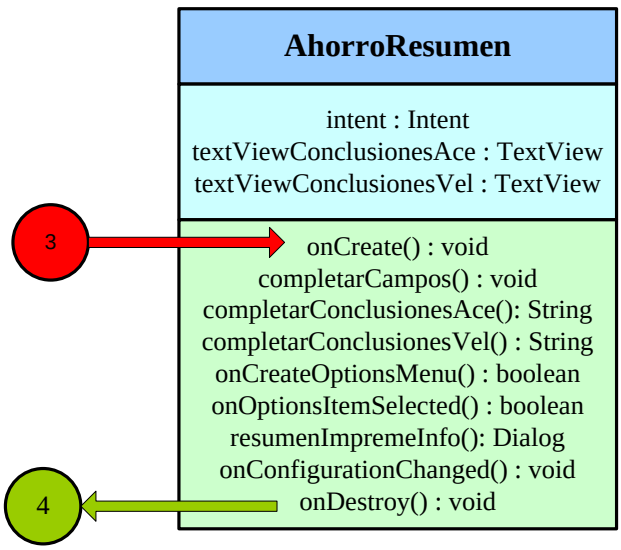


Figura 3.5. Diagrama de clases 3/3

La **figura 3.5** muestra la última parte del diagrama de clases de nuestra aplicación. En este caso, se trata de la clase ***AhorroResumen***, esta clase hereda de ***Activity***, por lo tanto, tiene su propia vista. El cometido principal de esta pantalla es mostrar un resumen con la información más importante relativa al viaje realizado, considerándose como importante, en este caso, aquella información que permita al usuario aprender de su actuación para poder mejorar en la habilidad de conducir eficientemente.

Como atributos, esta clase tiene dos objetos de tipo ***TextView*** que utilizaremos para generar unos mensajes de texto que muestren un diagnóstico de la conducción en un parámetro concreto de la misma. Además, esta clase tiene otro atributo, de tipo ***Intent***, que se llama ***intent*** y que almacena dicho objeto recibido de la actividad ***AhorroCombustible***, este objeto almacena toda la información que necesitamos para completar nuestra pantalla.

Se puede observar en la **figura 3.5** que esta clase se relaciona con la clase ***AhorroCombustible*** mediante los métodos comúnmente utilizados para este cometido. Cuando ***AhorroCombustible*** lance esta actividad se llamará al método ***onCreate()***, cuando abandonemos esta actividad se invocará al método ***onDestroy()*** y se devolverá el foco de atención a la actividad anterior.

3.2.3 Estructura del proyecto Android

A continuación, vamos a mostrar y a describir cómo se distribuyen los diferentes archivos que conforman la aplicación dentro de la estructura de carpetas. Esto nos permitirá de una manera sencilla ubicar cada uno de los archivos citados en los apartados posteriores.

Figura 3.6. Estructura de carpetas proyecto Android

Como podemos observar en la **figura 3.6**, el proyecto se llama **AhorroCombustible**. Este proyecto está dividido en 5 carpetas (***src, gen, assets, bin y res***) y tres archivos. Particularmente importante es el archivo ***AndroidManifest.xml***, se podría considerar el archivo principal de la aplicación, en apartados posteriores describiremos su contenido en este proyecto.

En la carpeta ***src*** se encuentra el código fuente Java almacenado dentro de un paquete, en este caso, el paquete se llama ***javier.pfc.ahorroCombustible***. Dentro de este paquete encontramos nuestras tres clases principales: **AhorroCombustible**, **AhorroConfiguracionPrevia** y **AhorroResumen**.

El directorio ***gen*** almacena archivos de código Java que se generan de forma automática a partir del contenido de la carpeta *res*. El archivo ***R.java*** sirve para indexar e identificar todos los recursos de la carpeta *res*.

Otra carpeta que aparece en el proyecto es la llamada ***assets***. Sirve para almacenar aquellos archivos que queramos que acompañen a la aplicación, serán empaquetados en el objeto "***.apk*****"** final. A diferencia de la carpeta *res*, los recursos que depositemos en esta carpeta no estarán indexados ni compilados. En nuestro caso, esta carpeta se encuentra vacía.

La carpeta ***bin*** se genera automáticamente al compilar, la utiliza el compilador como paso intermedio para preparar los archivos antes del empaquetado final (***.apk***).

Por último, encontramos el directorio ***res***. Esta carpeta contiene los recursos de la aplicación, se subdivide en múltiples subdirectorios que representan diferentes tipos de recursos. A continuación, describiremos la estructura interna de esta carpeta.

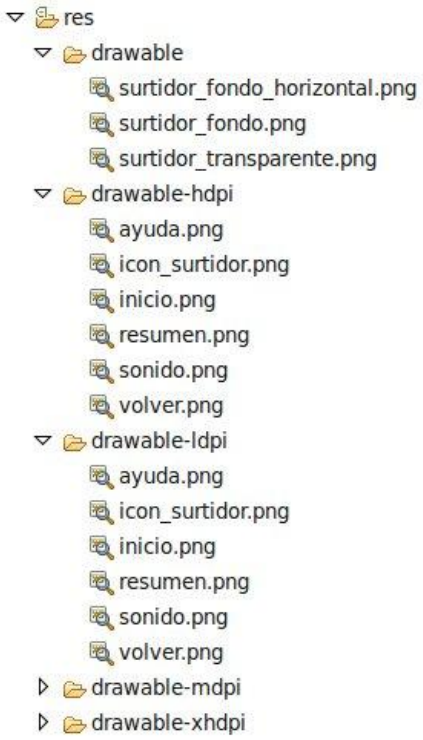


Figura 3.7. Contenido carpetas drawable

Como podemos observar en la **figura 3.7**, dentro de la carpeta ***res*** encontramos el directorio ***drawable***. Este directorio sirve para almacenar las imágenes que serán utilizadas en la aplicación. Hay varios directorios cuyo nombre se compone por ***drawable*** seguido de un sufijo, sirven para definir la misma imagen con distintos tamaños. Cada uno de los directorios tiene predefinido un tamaño de imagen, esto servirá para que se muestre la imagen de dimensiones adecuadas en función del tamaño y la densidad de la pantalla del dispositivo en el que esté corriendo la aplicación.

En nuestro caso, en el directorio llamado ***drawable*** (sin sufijos) guardamos las imágenes que se mostrarán en segundo plano (como fondo de la aplicación) y una imagen que adorna la pantalla principal. Estas imágenes son meramente decorativas por lo que se considera tolerable una posible pequeña pérdida de calidad en función de la pantalla del dispositivo en el que se ejecute la aplicación. Su tamaño es relativo al tamaño de la pantalla, por tanto, en pantallas grandes se ampliarán estas imágenes y en pantallas más pequeñas sus dimensiones se verán reducidas.

En el caso de los iconos utilizados en la aplicación, se considera importante que se adapten a la densidad de la pantalla para que mantengan siempre la misma apariencia. Debido a esto, se han establecido los mismos iconos con diferentes tamaños en cada una de las carpetas ideadas para este fin. En la carpeta ***drawable-xhdpi*** se ha depositado el más grande, en la carpeta ***drawable-ldpi*** el más pequeño, siguiendo las reglas establecidas por Android al respecto.

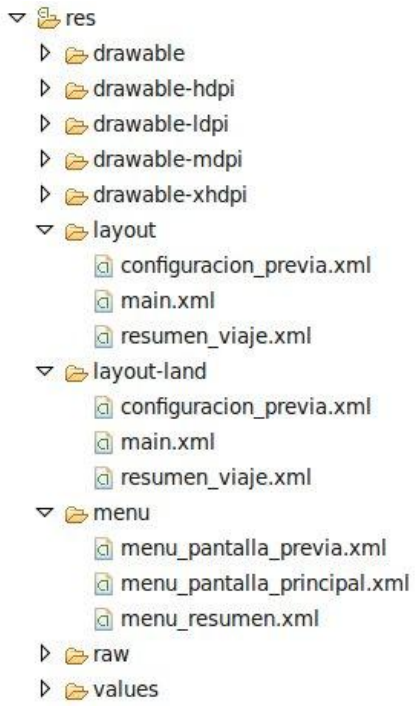


Figura 3.8. Contenido de las carpetas layout y menu

Dentro de la carpeta ***res*** encontramos también dos carpetas llamadas ***layout*** y ***layout-land***. En estas carpetas se guardan los ficheros XML que sirven para definir las interfaces de usuario. La carpeta ***layout*** guarda los ficheros XML que definen las interfaces que se mostrarán cuando el dispositivo se encuentre en posición vertical, por otro lado, la carpeta ***layout-land*** contiene los ficheros necesarios para generar las vistas cuando el dispositivo se encuentre en posición horizontal. Los ficheros contenidos en una carpeta tienen el mismo nombre y contienen los mismos elementos, aunque distribuidos de manera diferente, que en la otra carpeta. En nuestro caso, ambas carpetas contienen tres ficheros (cada una) que se corresponden con las vistas de las tres actividades que componen nuestra aplicación.

Debajo de las carpetas de vistas encontramos la carpeta ***menu***, que también se encuentra dentro de ***res,*** como muestra la **figura 3.8**, contiene los ficheros XML necesarios para generar los menús de las actividades. En nuestro caso, también hay un menú definido para cada actividad.

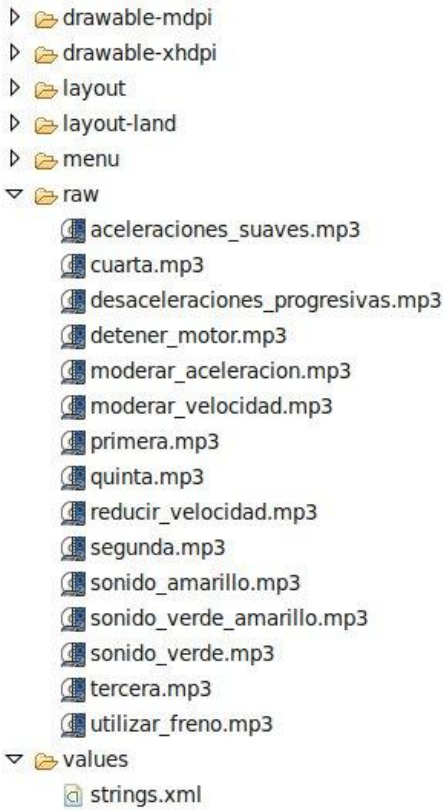


Figura 3.9. Contenido de las carpetas raw y values

Siguiendo con los directorios contenidos en la carpeta ***res***, en la **figura 3.9** encontramos el contenido del directorio ***raw***. Esta carpeta se utiliza para almacenar el resto de recursos que no están definidos en las otras carpetas. En nuestro caso, utilizamos esta carpeta para almacenar los ficheros de audio en formato "*MPEG* ("*Moving Picture Experts Group*") *Audio Layer III*" (mp3**)** que utilizará nuestra aplicación.

Por último, encontramos el directorio llamado ***values***. Esta carpeta almacena ficheros XML que pueden contener cadenas de texto (**strings**), colores o estilos. En nuestro caso, existe un único fichero que almacena todas las cadenas de texto que se muestran en los distintos interfaces de usuario de la plataforma.

3.2.4 Implementación de la aplicación AhorroCombustible

En esta sección describiremos la funcionalidad, entradas, salidas, etc., de los módulos y sub-módulos que conforman nuestra aplicación. También se incluirán los principales algoritmos desarrollados, explicados mediante algunos ejemplos.

3.2.4.1 El archivo manifiesto

Comenzaremos a analizar más en profundidad nuestra aplicación por el que podríamos considerar como el archivo principal de toda aplicación Android, el archivo ***AndroidManifest.xml***. Este es el contenido de dicho archivo en nuestra aplicación:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="javier.pfc.ahorroCombustible"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk  android:minSdkVersion="1"
               android:targetSdkVersion="15"  />

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <application
        android:icon="@drawable/icon_surtidor"
        android:label="@string/app_name"
        android:debuggable="true" >
        <activity
            android:name=".AhorroConfiguracionPrevia"
            android:configChanges="orientation|screenSize" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".AhorroCombustible"
            android:configChanges="orientation|screenSize" >
        </activity>
        <activity
            android:name=".AhorroResumen"
            android:configChanges="orientation|screenSize" >
        </activity>
    </application>
</manifest>
```

Código Fuente 1. AndroidManifest.xml

Comenzaremos a analizar el contenido de este archivo (mostrado en el **código fuente 1**) comentando el elemento ***<uses-sdk />*.** Este elemento contiene los atributos ***android:minSdkVersion*** y ***android:targetSdkVersion***. El primero de ellos indica el nivel mínimo del API de Android que necesita nuestra aplicación para correr (no podrá ser instalada en aquellos dispositivos que tengan instalado una versión anterior de Android), nuestra aplicación posee un código fuente sencillo y sólo utiliza librerías incluidas en el API 1.

El segundo elemento (***android:targetSdkVersion***) indica la versión de Android hacia la que va dirigida la aplicación, es decir, un dispositivo que tenga instalado el API 15 mostrará esta aplicación con el aspecto relativo a ese API, un dispositivo con un API 6 instalado mostrará la aplicación con la apariencia especifica de ese API (la apariencia de la aplicación varía si el API incluye modificaciones que afectan a la apariencia, por ejemplo, un cambio en la forma en la que se muestran los seekbar). En este caso, el número 15 en el atributo ***android:targetSdkVersion*** indica que la aplicación ha sido probada en ese API y todos los anteriores.

El elemento ***<uses-permission />*** incluye los permisos que serán solicitados al usuario cuando instale la aplicación. En este caso, solicitará permisos para acceder a la localización del dispositivo.

Dentro del elemento ***<application />*** se definen las actividades que componen nuestra aplicación. Además, debemos destacar el atributo ***android:debuggable***, este tiene definido el valor ***true***, fijando este valor podremos depurar nuestra aplicación en un dispositivo móvil a través del puerto USB. Este elemento es útil solamente durante el desarrollo, antes de generar el archivo "**.apk**" que utilizaremos para publicar nuestra aplicación en "Google Play" este elemento será eliminado.

Cada una de las actividades se identifica por un nombre, este se corresponde con el nombre de la clase en la que están definidas. En nuestro caso, es un nombre relativo al nombre del paquete descrito dentro del elemento ***<manifest />***.

El atributo ***android:configChanges*** evita que la actividad se reinicie cuando se produzca un cambio en la configuración, siempre que este se encuentre definido en la lista. En vez de reiniciarse, cuando uno de estos cambios tenga lugar se llamará al método ***onConfigurationChanged()***. Los dos elementos que contiene nuestra lista hacen referencia a un cambio en la configuración producido por la rotación del dispositivo (pasar de horizontal a vertical o viceversa), el elemento ***screenSize*** se ha de incluir en esta lista para niveles del API superiores al 13 ya que un cambio en la orientación del dispositivo implica un cambio en la relación de aspecto de la pantalla a partir de dicho API. **[17]**

El elemento ***<intent-filter />*** contiene el tipo de *intents* a los que responderá la actividad. En nuestro caso, los elementos contenidos indican que la actividad ***AhorroConfiguracionPrevia*** será la actividad que se lance cuando se inicie la aplicación.

3.2.4.2 Vistas de la aplicación

En este apartado, vamos a ver como se construyen los archivos XML que forman las vistas de la aplicación y los menús incluidos en las actividades.

Comenzaremos por algunos fragmentos del archivo ***configuracion_previa.xml*** (mostrados en el c**ódigo fuente 2**), incluido en la carpeta **/res/layout/**, este archivo conforma la vista de la pantalla de configuración, nos servirá a modo de ejemplo ya que el resto tienen una estructura muy similar.

```
<?xml version="1.0" encoding="UTF-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/surtidor_fondo" >
```

```
    <TextView
         android:id="@+id/previoLabel1"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/previo_instrucciones"
         android:layout_marginTop="2dip"
         android:layout_marginLeft="10dip"
         android:textColor="#FFFFFF" />

    <View
         android:id="@+id/previoSeparador1"
         android:layout_width="wrap_content"
        android:layout_height="2dip"
         android:layout_below="@id/previoLabel1"
         android:background="#FF909090"
         android:layout_marginTop="2dip"
         android:layout_marginBottom="2dip" />

(…)

    <SeekBar
         android:id="@+id/aceSeekBar1"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_marginTop="10dp"
         android:layout_marginLeft="10dip"
         android:layout_marginRight="10dip"
         android:max="50"
         android:layout_below="@id/textTraduceAceSeekBar1" />

(…)

    <Button
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text="@string/boton_comenzar"
         android:onClick="configurar"
         android:layout_below="@id/previoSeparador3"
         android:layout_centerHorizontal="true"
         android:layout_marginLeft="30dip"
         android:layout_marginRight="30dip" />

</RelativeLayout>
```

Código Fuente 2. configuracion_previa.xml

El elemento principal que conforma la pantalla es el llamado ***<RelativeLayout />***, se trata de un elemento que nos permite ubicar todos los objetos que van a formar nuestro *layout* de forma relativa, unos respecto a otros. Como observamos en sus atributos, esta vista va a ocupar toda la pantalla y tiene definida una imagen de fondo. Esta primera parte es común a todos los archivos de vistas generados para esta aplicación.

<TextView /> se utiliza para imprimir una etiqueta de texto en la pantalla. Uno de sus atributos es un identificador (***id***), se trata de un identificador único, necesario para todos los elementos que vayamos a referenciar, ya sea desde el código fuente o desde el mismo archivo XML. El atributo ***android:text*** determina el texto por defecto que va a mostrar dicha etiqueta. El resto de atributos hacen referencia a la estética (tamaño, márgenes, color).

El siguiente elemento que encontramos en el **código fuente 2** es ***<View />***. Este elemento es útil para dibujar en la pantalla. En nuestro caso, utilizamos este elemento para pintar líneas blancas que servirán de separadores entre distintas zonas de la pantalla. Cabe destacar el atributo ***android:layout_below***, este atributo fija la posición de este objeto debajo del indicado, referenciado mediante su identificador.

Siguiendo el contenido del archivo encontramos ***<SeekBar />***, se trata de una barra de progreso cuyo valor podemos modificar actuando sobre ella. Nuestra pantalla cuenta con cuatro objetos de este tipo que utilizamos para configurar los límites de velocidad y aceleración. Sus atributos nos indican que su valor será un número comprendido entre 0 y 50, ambos incluidos, aunque el mínimo no está definido explícitamente el valor por defecto para el mínimo es el 0. El rango de valores posibles es diferente en el caso de las barras definidas para la aceleración y las que se han definido para la velocidad.

El elemento ***<Button />*** pinta un botón en la pantalla. Cuando el usuario lo pulse se producirá una acción. A través del atributo ***android:onClick*** definimos el método de la clase asociada a este *layout* que será llamado cuando el usuario pulse el botón.

La forma en la que una actividad muestra un determinado *layout*, definido en la carpeta correspondiente, es a través de la siguiente línea:

```
setContentView(R.layout.nombre_layout);
```

Normalmente, este código se encuentra en el método ***onCreate()*** de la actividad, siempre se incluirá este código en la clase donde queramos que se muestre esta vista.

3.2.4.3 Menús de la aplicación

A continuación, vamos a describir de manera sencilla los pasos más importantes para generar y mostrar un menú en una actividad.

```
<?xml version="1.0" encoding="UTF-8"?>

    <menu xmlns:android="http://schemas.android.com/apk/res/android">

        <item android:id="@+id/resumen_id"
           android:icon="@drawable/resumen"
           android:title="@string/resumen"
           android:showAsAction="never" />

        <item android:id="@+id/inicio_id"
           android:icon="@drawable/inicio"
           android:title="@string/inicio"
           android:showAsAction="never" />

        <item android:id="@+id/sonido_id"
          android:icon="@drawable/sonido"
           android:title="@string/sonido"
           android:showAsAction="never" />
```

```
    <item android:id="@+id/ayuda_id"
       android:icon="@drawable/ayuda"
       android:title="@string/ayuda"
        android:showAsAction="never" />

</menu>
```

Código Fuente 3. menu_pantalla_principal.xml

En el c**ódigo fuente 3** se muestra el contenido completo del archivo ***menu_pantalla_principal.xml***. Dicho archivo compone el menú de la pantalla central de la aplicación, la pantalla creada por la actividad ***AhorroCombustible***.

Cada uno de los elementos ***<item />*** define un objeto añadido al menú. En este caso, se trata de cuatro elementos. Cada elemento tiene un identificador asociado, un título y un icono propio que se mostrará al desplegar el menú.

El atributo llamado ***android:showAsAction*** define como se va a mostrar dicho elemento en el caso de que el menú tenga forma de ***ActionBar***, en nuestra aplicación, se ha establecido el valor "**never**" para este atributo, esto quiere decir que todas las opciones del menú se mostrarán como parte del menú de **overflow**, o lo que es lo mismo, para ver las opciones disponibles habrá que desplegar el ***ActionBar*** mediante el elemento establecido para ello situado en la parte superior derecha de la pantalla.

ActionBar es la forma en la que Android aconseja mostrar los menús, pero sólo está disponible a partir del nivel API 11. Ya que en el atributo ***android:targetSdkVersion*** del archivo manifiesto establecimos el nivel 15, todos los dispositivos con un API 11 o superior instalado mostrarán los menús en forma de *ActionBar* y los que tengan APIs anteriores mostrarán el menú de la manera tradicional.

En el **código fuente 4** vamos a mostrar un fragmento de la clase ***AhorroCombustible.java*** que se encarga de crear el menú y de atender los eventos lanzados por este.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_pantalla_principal, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.resumen_id:
       lanzarActividadResumen();
        return true;
    case R.id.inicio_id:
       //Volvemos a la pantalla de inicio
       finish();
        return true;
    case R.id.ayuda_id:
       //Imprimimos la ayuda
       Dialog ayuda = imprimeAyuda();
       ayuda.show();
```

```
            return true;
        case R.id.sonido_id:
           //Activamos/Desactivamos sonido
           activarDesactivarSonido();
           return true;
        default:
            return super.onOptionsItemSelected(item);
        }
    }
```

Código Fuente 4. Fragmento AhorroCombustible.java

El primer método se encarga de generar el menú, este método es lanzado cuando el usuario pulsa el botón "menú" de su dispositivo o bien cuando el usuario pulsa sobre el *ActionBar*. El segundo método se encarga de atender los eventos lanzados por el menú, realiza las acciones pertinentes según el elemento seleccionado por el usuario.

3.2.4.4 Algoritmo de control de las barras de búsqueda de progreso

Como se ha comentado en apartados anteriores, en la pantalla inicial de la aplicación el usuario puede configurar los límites de la conducción, tanto de aceleración como de velocidad, que desea respetar durante el viaje.

Se han establecido cuatro valores configurables en nuestra aplicación, dos para la aceleración y dos para la velocidad. Tanto para la velocidad como para la aceleración existe un límite recomendable y un límite máximo. El límite recomendable marcará la franja en la que consideramos que nuestro vehículo está haciendo un uso eficiente del combustible. La aplicación considerará que se está realizando una conducción eficiente cuando nos encontremos por debajo de este límite, no obstante, será tolerante cuando este se supere levemente, aunque indicará al usuario tal situación.

El límite máximo significa aquella frontera que no queremos cruzar bajo ningún concepto, ya que circular por encima de este límite supone un gasto de combustible intolerable. Cuando superemos este límite la aplicación lanzará avisos para que el usuario corrija esa situación. Los avisos son más contundentes en este caso pues pretenden conseguir que el usuario reaccione ante ellos para que cese la situación actual. Este tipo de avisos no sólo se emplearán después de superar este límite, se empezarán a lanzar cuando superemos el punto medio entre el límite recomendable y el límite máximo.

En el caso de la aceleración, los límites se emplean tanto para la aceleración como para la deceleración (frenadas, reducciones de velocidad). El valor para los límites elegido se empleará para las aceleraciones, el mismo valor con un signo negativo delante se empleará para las deceleraciones.

Se han establecido valores por defecto para los cuatro límites. En el caso de la velocidad, el valor escogido para el límite recomendable es de 60 Km/h, este valor implica que circulando en 5ª marcha estamos en la zona de menor consumo de nuestro vehículo, en el menor régimen de rpm posible. Este límite, además, nos permite circular

por una vía urbana utilizando los límites establecidos por la autoridad competente. Este valor, sin embargo, es bajo cuando circulamos por una autovía pero la tolerancia de la aplicación nos permite circular hasta los 79 Km/h sin considerar un incumplimiento grave de los límites. El valor del límite máximo se ha establecido en 100 Km/h, se ha elegido ese valor ya que es el límite de velocidad en muchas vías y, además, según la **figura 2.7** es la franja de velocidad sobre la que el consumo se dispara.

Los límites por defecto para la aceleración se han fijado en 0.9 y 1.3 m/s^2, para la aceleración recomendable y la aceleración máxima respectivamente. Un turismo medio posee una aceleración máxima de 2.5 m/s^2, se ha convenido para el límite máximo la mitad de dicho valor. Para el límite recomendable se ha considerado un límite que no sea superado en el caso de que el vehículo desacelere empleando el freno motor, que se ha observado que significa una desaceleración de -0.6 m/s^2 aproximadamente.

Por otro lado, es muy importante, en la parte de una aplicación en la que esta solicita información al usuario, cerciorarse de que este no compromete la integridad de la aplicación a través de su actuación. En nuestra aplicación, el usuario puede introducir una serie de valores sobre los que la aplicación va a trabajar. Los valores que puede establecer el usuario han de cumplir una serie de requisitos establecidos por la aplicación. En este caso:

- Los valores han de ser completamente numerales (no pueden contener otros caracteres).
- No se permiten números negativos.
- En el caso de la aceleración, el dato consistirá en un número comprendido entre 0 y 10’1, ambos incluidos, con una precisión de un decimal.
- En el caso de la velocidad, el dato consistirá en un número entero comprendido entre 0 y 220, ambos incluidos.
- En ningún caso el valor de los límites máximos puede ser igual o inferior al valor de los límites recomendables establecidos.

Para asegurar que el usuario cumple dichos requisitos en los datos de entrada, se ha establecido que este introduzca dichos valores a través de las barras de búsqueda de progreso (*SeekBars*), se han incluido un total de 4 objetos tipo *SeekBar* dentro de la clase ***AhorroConfiguracionPrevia***, dos para los límites de la aceleración y dos para los límites de la velocidad. Mediante estas barras, previamente, establecemos los límites necesarios para un correcto funcionamiento de la aplicación y así evitamos pesados códigos que comprueben cada dato introducido.

El progreso de los objetos tipo *Seekbar* consiste en un número entero comprendido entre los límites establecidos, en este caso, dichos límites se encuentran en la declaración de cada uno de los objetos dentro del archivo de vistas ***configuración_previa.xml***.

En el caso de las barras de progreso destinadas a configurar la aceleración, el intervalo de valores que el progreso puede tomar comprende desde el 0 hasta el 50,

ambos incluidos. En realidad, tomaremos como valor para la aceleración ese número dividido entre 10, debido a esto, el margen seleccionable para dicha aceleración comprenderá desde 0,0 hasta 5,0. Las unidades empleadas para medir la aceleración son **metros/segundo^2**. Se han escogido estos límites como razonables ya que la aceleración máxima establecida (5,0 m/s^2) implica que el vehículo tardaría en alcanzar los 100 km/h partiendo del reposo aproximadamente 5 segundos, esta capacidad de aceleración está al alcance solamente de los vehículos destinados a la competición.

Por otro lado, cabe destacar que los valores que se van a mostrar en la pantalla de la aplicación no se corresponden exactamente con el progreso seleccionado por el usuario, esto ocurre en el caso de la barra destinada a fijar el límite máximo. Partimos de la condición de que el límite máximo ha de ser siempre superior al límite recomendable. Se ha establecido, por tanto, que el rango de valores seleccionables de la segunda barra empiece desde el valor seleccionado en la primera, de este modo, si el usuario en la barra de búsqueda de progreso dedicada al límite recomendable de la aceleración selecciona el valor 2,0 m/s^2, por ejemplo, en la segunda barra (la destinada al límite máximo) podrá seleccionar el rango de valores que empieza en 2,1 y acaba en 7,1 m/s^2. El límite superior es establecido como resultado de sumar los 50 valores posibles de progreso al seleccionado en la primera barra, en el caso del ejemplo será 20.

Por otro lado, en las barras de búsqueda de progreso destinadas a seleccionar los límites de la velocidad sí que vamos a trabajar con números enteros. En el caso de la barra destinada al límite recomendable, se ha establecido el rango de valores que abarca desde 0 hasta 120. En este caso trabajamos con **km/h**, de modo que el usuario en la primera barra puede seleccionar desde 0 hasta 120 km/h.

En el caso de la segunda barra, el rango de valores empieza en 0 y acaba en 100. Igual que ocurría con la aceleración, en la pantalla se mostrará como valor mínimo seleccionable el valor establecido en la primera barra sumándole 1, el valor máximo seleccionable será el valor mínimo más 100. Por ejemplo, si el usuario selecciona como límite recomendable el valor 60 km/h, en la barra del límite máximo podrá establecer un valor comprendido dentro del rango que abarca desde 61 hasta 161 km/h.

A continuación, vamos a mostrar y describir, mediante su pseudocódigo, como gestiona la aplicación estas barras de progreso, esta gestión asegura datos consistentes con los requisitos descritos anteriormente. Comenzaremos por los atributos de la clase ***SeekBarListener*** empleados en este algoritmo, estos atributos se muestran en el **código fuente 5**:

```
TextView textProgressAce1;
TextView textProgressAce2;
TextView textProgressVel1;
TextView textProgressVel2;

Integer progressAce1;
```

```
Integer progressAce2;
Integer progressVel1;
Integer progressVel2;
```

Código Fuente 5. Atributos asociados a las Seekbar de la clase SeekBarListener

Los primeros cuatro atributos, mostrados en el **código fuente 5,** son objetos que forman parte de la interfaz de usuario, en ellos mostraremos el valor escogido por el usuario en la barra de progreso. El resto de atributos almacenan el valor entero devuelto por una barra de progreso, cada uno de los cuatro pertenece a una *SeekBar* diferente.

Los sufijos "***Vel***" indican que dicho atributo se corresponde con un valor que afecta a la velocidad, los sufijos "***Ace***", sin en cambio, referencian los valores de la aceleración. Por su parte, los sufijos "***1***" y "***2***" indican si el atributo se utiliza para el valor máximo (en cuyo caso corresponde el sufijo "*2*") o, por el contrario, se utiliza para el valor recomendable (sufijo "*1*").

Cada una de nuestros objetos *Seekbar* tiene asignado un escuchador, en nuestro caso, las cuatro barras comparten el mismo escuchador (un escuchador es un objeto de la clase ***SeekBarListener***). Cuando el usuario actúa sobre una de las barras de progreso se lanza automáticamente el método ***onProgressChanged*()**, implementado en la clase del escuchador. Este es su pseudocódigo (**código fuente 6**):

```
void onProgressChanged(SeekBar seekBar, Integer progress) {

    Integer idSeekBar = seekBar.getId();

    if(idSeekBar == R.id.aceSeekBar1) {

        gestionaAceSeekBar1(progress);

    } else if(idSeekBar == R.id.aceSeekBar2) {

        gestionaAceSeekBar2(progress);

    } else if(idSeekBar == R.id.velSeekBar1) {

        gestionaVelSeekBar1(progress);

    } else {

        gestionaVelSeekBar2(progress);
    }
}
```

Código Fuente 6. Método onProgressChanged() de la clase SeekBarListener

Este método no devuelve ningún valor y recibe como argumentos, que sean relevantes para nosotros, un objeto de la clase *Seekbar* y un número entero con el valor del progreso de dicho objeto.

El objetivo del método es identificar aquella de las barras que lanzó el evento, discriminando estas por su identificador (Id), después, una vez averiguado esto, llama al método correspondiente que se va a encargar de gestionar como se va a mostrar el

resultado en la pantalla de la actividad. En la llamada al segundo método, incluye el progreso de la barra con la que estamos trabajando.

Para llevar a cabo la funcionalidad anterior, el método comienza obteniendo el identificador de la barra que ha lanzado este y, después, lo compara con los identificadores de las barras definidas en el proyecto. Estos identificadores están almacenados en la clase ***id***, la cual, es una clase anidada a la clase ***R***.

A continuación, vamos a mostrar el pseudocódigo (**código fuente 7**) de los métodos que son llamados desde el método *onConfigurationChanged()*. Comenzaremos por el método ***gestionaAceSeekBar1()***:

```
void gestionaAceSeekBar1(int progress) {

    progressAce1 = progress;

    float progressFloatAce1 = (float)progressAce1 / 10;
    float progressFloatAce2 = 0;

    textProgressAce1.setText(Float.toString(progressFloatAce1));

    if (progressAce1 >= progressAce2) {
        if(progressAce1 == 0) {
            progressFloatAce2 = progressFloatAce1;
        } else if (progressAce2 == 0) {
            progressFloatAce2 = (float)(progressAce1+1) / 10;
        } else {
            progressFloatAce2 =
                (float)(progressAce2+progressAce1) / 10;
        }

            textProgressAce2.setText(
        Float.toString(progressFloatAce2));

        }
    }
```

Código Fuente 7. Método gestionaAceSeekBar1() de la clase SeekBarListener

El código de este método es un poco más complejo que el anterior, debido a esto, iremos describiendo el mismo paso a paso. Para empezar, este método se encarga de atender los eventos de la barra que establece el límite recomendable de la aceleración. En la declaración del método observamos que recibe el valor del progreso de la barra en forma de número entero.

En la primera línea, actualizamos el valor del atributo de la clase ***SeekBarListener***, correspondiente a esta barra, con el valor recibido como argumento.

A continuación, creamos dos variables locales para este método, ambas serán números con coma flotante (números con decimales). La primera de las variables es el valor recibido dividido entre diez, esto se debe a que el valor del progreso es un número entero, en este caso, dicho valor se encuentra definido entre 0 y 50 (ambos incluidos), para obtener nuestro valor de aceleración dividimos este número entre 10, así

obtenemos un valor de aceleración comprendido entre 0 y 5 con un decimal. Por ejemplo, si el progreso recibido fuese 33, el valor de la aceleración que utilizaríamos sería 3,3. Estas variables nos servirán para actualizar el valor de las etiquetas de la aceleración en la pantalla de la actividad.

En la siguiente línea se actualiza el contenido de la etiqueta asociada a la barra de progreso con la que estamos trabajando, se muestra en esta etiqueta el valor de la aceleración que hemos calculado, aquel que contiene un decimal.

El resto del método se completa con una estructura condicional "***if***". El código contenido en este condicional se ejecuta siempre que el valor de la segunda barra de progreso (aquella que fija el límite máximo para la aceleración) sea igual o inferior que el límite recomendable de la aceleración. Sería una incongruencia que el límite recomendable fuese mayor que el máximo, del mismo modo, se ha establecido que ambos límites no pueden ser iguales. Cuando una de estas condiciones ocurra se ejecutará el código contenido en este bloque.

Dentro del bloque de código citado en el apartado anterior (bloque ***if***), se establece un valor consistente con el valor de la barra encargada del límite recomendable en la barra de progreso que se encarga de gestionar el límite máximo de la aceleración. Es importante destacar que los valores que vamos a modificar son los valores contenidos en las etiquetas asociadas que muestran dichos valores, en ningún caso, se modifica el valor de progreso de la propia barra. Se han establecido tres casos en los que el valor de la segunda barra será modificado en función del valor de la primera barra.

El primero de los casos establecidos comprueba que el valor de la primera barra de progreso (límite recomendable) sea cero, teniendo en cuenta la condición superior (que el progreso de la primera sea mayor o igual que el de la segunda) habremos entrado aquí si el progreso de la segunda barra también es cero, en ese caso, se actualiza el valor de la etiqueta asociada a la segunda barra para que esta también muestre el valor cero, concretamente mostrará el valor 0,0.

En el segundo caso, si el progreso de la segunda barra (límite máximo) es cero y el progreso de la primera no lo es, modificamos el valor de la etiqueta asociada a la segunda barra con el resultado de sumarle 0,1 al progreso de la primera barra dividido entre 10. Por ejemplo, si el progreso de la primera es 21 y el progreso de esta segunda barra es 0, en la etiqueta asociada a la primera barra se mostrará el valor 2,1 y en la etiqueta asociada a la segunda barra mostraremos el valor 2,2.

En el último caso se contempla la única posibilidad restante, es decir, que ambos progresos sean distintos de cero y que el primero de ellos sea igual o superior al segundo, como consecuencia de la condición que engloba a los tres casos. Si se cumplen estas condiciones actualizaremos el valor de la etiqueta asociada a la segunda barra de progreso con el resultado, dividido entre diez, de sumar los progresos de ambas barras. Por ejemplo, si el progreso de la primera es 33 y el progreso de la segunda es 22,

en la etiqueta de la segunda barra se mostrará 5,5, en la primera etiqueta mostraríamos 3,3.

En la última línea del método ***gestionaAceSeekBar1()*** se actualiza el valor de la etiqueta asociada a la segunda barra de búsqueda de progreso con el resultado obtenido de uno de los casos anteriores.

Una vez terminada la descripción del primero de los métodos que son llamados desde el método ***onProgressChanged()*** (todos ellos incluidos en la clase ***SeekBarListener***), en función de la barra que lance el evento, vamos a continuar con el segundo de estos métodos, el método llamado ***gestionaAceSeekBar2()***, este es su pseucódigo (**código fuente 8**):

```
void gestionaAceSeekBar2(int progress) {

    float progressFloatAce2 = 0;
    progressAce2 = progress;

    if(progressAce1 != 0) {
           progressFloatAce2 =
        (float)(progressAce2+progressAce1+1) / 10;
    } else {
           progressFloatAce2 =
        (float)progressAce2 / 10;
    }

    textProgressAce2.setText(Float.toString(progressFloatAce2));

}
```

Código Fuente 8. Método gestionaAceSeekBar2() de la clase SeekBarListener

Este método se encarga de atender los eventos generados por la segunda barra definida para la aceleración, aquella que se encarga de fijar el límite máximo para la aceleración, recibe como argumento el valor del progreso de dicha barra en forma de número entero. Como se puede observar en el **código fuente 8**, el código de este método es más sencillo que el método descrito anteriormente, esto es debido a que el valor seleccionado por el usuario, en esta segunda barra, nunca va a modificar el valor mostrado en la etiqueta asociada a la primera barra, al contrario de lo que ocurría en el primer método.

En este caso, en el método sólo se define una variable local que utilizaremos para calcular el valor que mostraremos en la etiqueta correspondiente. En la segunda línea se actualiza el valor del atributo que contiene el progreso de esta segunda barra de búsqueda de progreso.

En las siguientes líneas se han establecido dos condiciones para calcular el valor de la etiqueta, que el progreso de la primera barra (límite recomendado) sea cero o distinto de cero. En el caso de que el progreso de la primera barra sea distinto de cero, el valor de la etiqueta asociada a la segunda barra será el resultado, dividido entre diez, de la suma de los dos progresos referentes a la aceleración más 1. La razón de sumar uno es

obligar así a que el valor de la etiqueta de la segunda barra sea siempre superior al valor de la etiqueta de la primera barra. Por ejemplo, si el valor de progreso de la primera es 15 y el valor de la segunda se ha establecido en 0, en la etiqueta de la segunda barra se va a mostrar el valor 1,6.

Por el contrario, si el valor del progreso de la primera barra es 0, el valor que vamos a mostrar en la etiqueta asociada a la segunda barra de búsqueda de progreso es, simplemente, el valor de progreso de esta barra dividido entre 10. Por ejemplo, si la primera barra muestra un 0,0 y en la segunda barra se selecciona el progreso 25, en la etiqueta asociada a esta segunda barra mostraremos el valor 2,5.

Al final de este método se actualiza el valor mostrado en la etiqueta asociada a la barra encargada de fijar el límite máximo tolerable para la aceleración con el valor calculado mediante este método.

Con el método anterior, hemos completado aquellos métodos que se encargan de los valores asociados a los límites de la aceleración. Vamos a continuar con los métodos encargados de gestionar los eventos procedentes de las barras de búsqueda de progreso asociadas a los límites para la velocidad. Como en los casos anteriores, un 1 al final del nombre de una variable o un atributo indica que está relacionado con el límite recomendado de la velocidad, por el contrario, un 2 al final de uno de esos nombres indica que estamos trabajando con el límite máximo de velocidad.

Vamos ahora a mostrar el pseudocódigo del método que recoge los eventos de la barra de búsqueda de progreso asociada al límite recomendable de la velocidad, el método se llama ***gestionaVelSeekBar1()*** y, como los anteriores, se encuentra dentro de la clase ***SeekBarListener***, su pseudocódigo se muestra en el **código fuente 9**:

```
void gestionaVelSeekBar1(int progress) {

    int progressIntVel2 = 0;

    progressVel1 = progress;
    textProgressVel1.setText(Integer.toString(progressVel1));

    if (progressVel1 >= progressVel2) {

        if(progressVel1 == 0) {
            progressIntVel2 = progressVel1;
        } else if (progressVel2 == 0) {
            progressIntVel2 = progressVel1 + 1;
        } else {
            progressIntVel2 = progressVel2+progressVel1;
        }

        textProgressVel2.setText(Integer.toString(progressIntVel2));
    }
}
```

Código Fuente 9. Método gestionaVelSeekBar1() de la clase SeekBarListener

Trabajar con velocidades es más sencillo que con aceleraciones ya que las velocidades están representadas por números enteros, esto implica que no hay que hacer conversiones, al contrario de lo que ocurría con la aceleración.

Todos los métodos relacionados con las barras de progreso reciben como argumento el número entero que representa el progreso de la barra con la que trabajamos. El método comienza declarando una variable local, como un número entero, que utilizaremos para modificar el texto incluido en la etiqueta relacionada con el límite máximo de velocidad, siempre que sea necesario hacerlo.

En la segunda línea actualizamos el valor del atributo de esta clase que almacena el progreso de esta primera barra asociada a la velocidad. En la tercera línea modificamos el valor de la etiqueta asociada a esta barra con el valor obtenido del parámetro de entrada del método.

A continuación, podemos observar de nuevo una estructura condicional que nos servirá para decidir en qué casos y de qué forma vamos a modificar el valor mostrado en la etiqueta asociada al límite máximo de la velocidad. Esta estructura condicional está englobada dentro de una comparación, la condición, en este caso, comprueba que el progreso de la barra 1 sea mayor o igual que el progreso de la barra 2. Al igual que ocurre con la aceleración, no tiene sentido un límite máximo menor o igual que el límite recomendable.

Dentro de esta condición que engloba a las demás se han definido tres casos, igual que con la aceleración, que el progreso de la barra 1 sea cero, que sea cero el progreso de la barra 2 y que ninguno de los dos lo sea. En el primer caso, la segunda etiqueta se pondrá a cero. En el segundo caso, el valor de la segunda etiqueta será el valor del progreso de la primera barra sumándole 1, por ejemplo, si el progreso de la segunda barra es 0 y el progreso de la primera es 50, en la segunda etiqueta (la asociada al límite máximo) se mostrará el valor 51. En el tercer caso, si ninguno de los progresos es 0, el valor mostrado en la segunda etiqueta será el resultado de la suma de los progresos de las barras asociadas a la velocidad.

Por último, nos queda describir un poco el método encargado de gestionar los eventos producidos por la segunda barra asociada a la velocidad, aquella que se encargar de establecer el límite máximo, este es su pseudocódigo, descrito en el **código fuente 10**:

```
private void gestionaVelSeekBar2(int progress) {

    int progressIntVel2 = 0;

    progressVel2 = progress;

    progressIntVel2 = progressVel2+progressVel1;

    if(progressVel1 != 0) {
        progressIntVel2++;
    }
```

```
        textProgressVel2.setText(Integer.toString(progressIntVel2));
    }
```

Código Fuente 10. Método gestionaVelSeekBar2() de la clase SeekBarListener

Este método se comporta de la misma forma que ***gestionaAceSeekBar2*()**, con la diferencia de que este último trabajaba con decimales por lo que su estructura cambia un poco. Comienza este método igual que el resto, definiendo una variable local que nos servirá en los cálculos y actualizando el valor del progreso en el atributo correspondiente.

En este caso, para cumplir la condición establecida, consistente en que el límite máximo sea siempre superior al límite recomendado, se ha establecido una sola condición. Para empezar, el valor que vamos a mostrar en la etiqueta asociada al límite máximo va a ser la suma de los progresos de las dos barras destinadas a configurar los límites de la velocidad. La condición a cumplir es que el progreso de la primera barra sea distinto de cero, en ese caso se añade 1 al resultado de la suma anterior, de esta forma evitamos que se muestre el mismo valor en ambas etiquetas (indicando que ambos límites serían iguales) cuando el progreso de la segunda barra sea cero y el de la primera barra no lo sea. Por ejemplo, si el progreso de la primera barra es 72 y el de la segunda es 0, en la segunda etiqueta se mostrara el valor 73, de esta manera, el límite recomendado sería 72 y el límite máximo sería 73.

3.2.4.5 Algoritmo de procesamiento de datos de localización

Como se ha comentado en anteriores ocasiones, el objetivo del presente proyecto es desarrollar una aplicación que pueda aconsejar al usuario en tiempo real acerca de actuaciones que le permitan conducir de manera eficiente, o lo que es lo mismo, ahorrar combustible durante el viaje. Para que la aplicación pueda llevar a cabo dicho objetivo necesita tener conocimiento de los datos del viaje en tiempo real, con objeto de proveer al sistema de dichos datos empleamos el dispositivo GPS instalado en el terminal.

El sistema GPS entrega datos referentes a la posición del dispositivo en forma de objetos de tipo ***Location***. Dentro de este objeto encontramos datos como latitud, longitud, precisión del dato, fecha de creación del dato, velocidad, altitud, rumbo, etc.

Mediante la siguiente línea de código mostrada en el **código fuente 11** y extraída del método ***onResume*()**, incluido en la clase **AhorroCombustible**, configuramos la recepción de los objetos descritos en el párrafo anterior, más concretamente, registramos el escuchador de eventos de ese tipo en el objeto encargado de gestionar la localización en nuestra aplicación:

```
mlocManager.requestLocationUpdates(

    LocationManager.GPS_PROVIDER, 0, 0, mlocListener);
```

Código Fuente 11. Configuración de recepción de datos GPS en AhorroCombustible

Los atributos *mlocManager* y *mlocListener* fueron mencionados en el apartado del diagrama de clases. El atributo *mlocManager* se encarga de manejar la configuración que nos va a permitir trabajar con los objetos *Location* necesarios. Por su parte, el atributo *mlocListener* es el encargado de atender a los eventos de localización generados por el dispositivo, es el escuchador de dichos eventos.

En la línea de código mostrada en el **código fuente 11** configuramos el manejador de localización para que reciba los objetos de localización del proveedor GPS del terminal. Los dos 0 indican que no se imponen restricciones, a priori, referentes al tiempo entre muestras recibidas ni a la distancia mínima entre ellas (descartaremos nosotros mismos aquellas coordenadas que se encuentren muy próximas entre sí, ya sea en distancia o en tiempo). Por último, registramos el escuchador de eventos que se va a encargar de recibir y gestionar los objetos *Location* enviados por el GPS. El escuchador de eventos es un objeto de la clase ***MyLocationListener*** que implementa la interfaz ***LocationListener***.

Cuando se lanza un evento de localización que implica un cambio en la posición del terminal se lanza automáticamente el método ***onLocationChanged*()** implementado en la clase del escuchador. Dicho método y los métodos asociados a este centran la funcionalidad principal de la aplicación, debido a esto, describiremos detalladamente el algoritmo incluido en este método. El algoritmo presente en este método es relativamente complejo por lo que trataremos de abstraernos de ciertos detalles irrelevantes acerca de cómo se reproduce cierto sonido o se pinta de un determinado color una etiqueta en concreto. Además, procuraremos separar el algoritmo general en los algoritmos más sencillos que lo componen.

Para hacernos una idea de la complejidad de este algoritmo basta con observar, en el **código fuente 12**, los atributos necesarios para llevar a cabo el mismo, incluidos en la clase del escuchador:

```
Location locationAnterior;

float distancia;
float tiempo;
float velocidad;
float aceleracion;
float altitud;
float factorPendiente;
float distanciaCalcularPendiente;
float distanciaCalcularPendienteNueva;
Vector<Float> vector_altitudes;
float precision;

boolean primeraCoordenada;
boolean segundaCoordenada;

TextView textViewAceleracion;
TextView textViewVelocidad;
TextView textPendiente;
TextView textVariacionConsumo;
TextView textViewMarcha;
```

```
ImageView imageViewBack;
TextView textViewLabelAceleracion;
TextView textViewLabelVelocidad;
TextView textViewConsejos;

float tiempoEntreNotificaciones;
float tiempoEntreNotificacionesMarcha;
float tiempoParada;
int gradoConsumo;

float masaPorGravedad;

Dialog estadoGPS;
```

Código Fuente 12. Atributos de la clase MyLocationListener

Describiremos la utilidad de cada uno de estos atributos a medida que vayan apareciendo en el algoritmo. Comencemos a mostrar el pseudocódigo incluido en el método ***onLocationChanged(Location loc)***, mediante el **código fuente 13**:

```
float precisionNueva = loc.getAccuracy();

if(precisionNueva > (float)25) {
   return;
} else if (precisionNueva > (precision + (float)15)) {
   return;
} else if (precisionNueva < precision) {
   precision = precisionNueva;
}
```

Código Fuente 13. Primera parte pseudocódigo del método onLocationChanged()

En este código hacemos la primera comprobación del dato de localización obtenido. En la primera línea extraemos la precisión del objeto recibido. En ningún caso vamos a tolerar una precisión mayor de 25 metros por lo que si ese es el caso descartamos esa coordenada.

Uno de nuestros atributos tipo coma flotante (float) llamado "**precision**" almacena el mejor valor de precisión obtenido de las coordenadas. Con esto queremos evitar que si estamos obteniendo coordenadas con una precisión de 15 metros se cuele una coordenada que tenga una precisión mucho peor que estas, ya que esto podría inducir un error en los cálculos de posición, velocidad, etc. En el segundo condicional comprobamos que la precisión de la coordenada recibida sea mayor que el mejor dato de precisión obtenido con anterioridad más 15 metros, es decir, si el mejor dato obtenido son 5 metros, no aceptaremos coordenadas con una precisión mayor a 20 metros.

En el último caso, si la precisión de la coordenada recibida es mejor que el dato almacenado con anterioridad nos quedaremos con el nuevo dato de precisión.

Para poder calcular la velocidad, el tiempo y la distancia necesitamos tener dos coordenadas consecutivas, para ello, se han creado los atributos ***locationAnterior*** y ***primeraCoordenada***. El primero de ellos almacena el anterior objeto Location recibido,

de esta forma, podremos hacer los cálculos a partir del objeto actual y del anterior. Por otro lado, el atributo **primeraCoordenada** funciona a modo de contador, por defecto su valor es "true" y cuando se acepta la primera coordenada (después de lanzar la actividad) se actualiza al valor "false". Así conseguimos empezar a hacer los cálculos cuando tenemos datos para ello. En el **código fuente 14** se muestra la estructura general que se encarga de comprobar esta condición:

```
void onLocationChanged(Location loc) {

    (…)

    if(!primeraCoordenada) {

        (…)

    } else {
        primeraCoordenada = false;
    }

    locationAnterior = loc;

    (…)
}
```

Código Fuente 14. Estructura general contador de coordenadas en onLocationChanged()

El símbolo (…) indica que en ese lugar hay código que no se muestra. El fragmento de código que falta antes del condicional "if" es el código encargado de comprobar la precisión de las muestras descrito anteriormente.

A continuación, vamos a seguir con el pseudocódigo de este método, continuaremos ahora con el fragmento de código que está contenido en el bloque "if", es decir, aquel que se ejecuta una vez recibida la segunda coordenada, podremos observarlo en el **código fuente 15**:

```
if(!primeraCoordenada) {

    //Obtenemos el tiempo transcurrido entre las dos últimas
    coordenadas
    long tiempoMilis = loc.getTime() - locationAnterior.getTime();

    //Tiempo entre coordenadas mínimo igual a un segundo
    if(tiempoMilis < (long)1000) {
          return;
    }

    tiempo = (float)(tiempoMilis / (long)1000);

    distancia = locationAnterior.distanceTo(loc);
    //Si la distancia entre coordenadas es menor de 1 m la
    consideramos 0
    if(distancia < (float)1) {
        distancia = 0;
    }

    (…)
```

```
} else {
    primeraCoordenada = false;
}
```

Código Fuente 15. Primeros cálculos en el método onLocationChanged()

Este primer fragmento de código, mostrado en el **código fuente 15**, es bastante sencillo, en él primeramente calculamos el tiempo entre las dos últimas coordenadas recibidas, en milisegundos. En la siguiente condición comprobamos que el tiempo transcurrido entre estas sea de al menos 1 segundo. Descartamos todas aquellas coordenadas que no cumplan esta condición. Con esta misma condición nos aseguramos de que no vamos a aceptar coordenadas más antiguas que la última recibida, esto es posible ya que una coordenada con una fecha anterior a la última almacenada daría como resultado de la primera operación un tiempo negativo, en ese caso, sería descartada por la condición anterior que implica un tiempo mínimo entre coordenadas mayor a 1 segundo.

En la siguiente sentencia actualizamos el valor del atributo tiempo, que nos va a servir para hacer cálculos de aceleraciones y velocidades, a partir del tiempo calculado anteriormente convirtiéndolo a segundos.

El método ***Location.distanteTo(Location)***, disponible en el API de Android, nos permite calcular la distancia, en metros, entre dos objetos *Location*, de esta forma podemos calcular la distancia entre las dos coordenadas recibidas.

En el caso de que la distancia resultante de la operación sea menor a 1 metro vamos a considerarla 0. La razón de considerar 0 todas las distancias menores a 1 metro viene dada por la propia precisión del GPS, en condiciones normales, permaneciendo con el GPS habilitado y sin movernos del sitio, la lectura de la posición ofrecida por el GPS varía dentro de ese margen. Se ha comprobado que permaneciendo el dispositivo inmóvil, el GPS entrega posiciones que varían entre 0 y 100 cm. Por eso, vamos a descartar esas distancias ya que para nuestra aplicación no suponen una pérdida de precisión apreciable. Una distancia de 1 metro entre dos coordenadas separadas por 1 segundo da como resultado una velocidad de 3,6 km/h (1 m/s), esta velocidad es demasiado baja para los vehículos hacia los que está destinada la aplicación.

Vamos a continuar ahora mostrando el fragmento de código restante hasta la sentencia “else” gracias al **código fuente 16**:

```
if(!primeraCoordenada) {

    (…)

    //Para que los valores de distancia, aceleración y velocidad
    sean consistentes cuando está parado
    if(distancia == 0) {

        aceleracion = 0;
        velocidad = 0;

        if(segundaCoordenada) {
```

```
                segundaCoordenada = false;
            }

        } else {

            //Calculamos el tiempo entre los avisos sonoros
            tiempoEntreNotificaciones += tiempo;
            tiempoEntreNotificacionesMarcha += tiempo;

            float velocidadActual = 0;

            if (loc.hasSpeed()) {
                velocidadActual = loc.getSpeed();
                validarVelocidadDistancia(velocidadActual);

            } else {
                velocidadActual = distancia / tiempo;
            }

            if(!segundaCoordenada) {

                aceleracion = (velocidadActual - velocidad) / tiempo;

                (…)

            } else {
                segundaCoordenada = false;
            }

            velocidad = velocidadActual;

            (…)
        }
    } else {
        primeraCoordenada = false;
    }
```

Código Fuente 16. Primeros cálculos, segunda parte, en el método onLocationChanged()

Como se puede observar, hay dos fragmentos del pseudocódigo que se han suprimido, estos dos fragmentos se encargan de hacer cálculos sobre datos que se van a mostrar en la pantalla resumen y que, por tanto, no son relevantes en este algoritmo.

En el pseudocódigo aparece una estructura condicional que engloba toda esta parte, tiene que ver con la distancia. En el caso de que la distancia obtenida entre las dos últimas coordenadas recibidas sea 0, tanto la velocidad como la aceleración han de serlo también.

Igual que ocurría con la primera coordenada también es importante esperar a recibir la segunda para realizar ciertos cálculos, es el caso de la aceleración, para calcular la aceleración necesitamos conocer dos velocidades y el tiempo transcurrido. La secuencia sería la siguiente:

1. Recibo primera coordenada: primeraCoordenada = false, velocidad = 0 y aceleración = 0
2. Recibo segunda coordenada: segundaCoordenada = false, calculamos velocidad y aceleración = 0
3. Recibo tercera coordenada y siguientes: puedo calcular velocidad y aceleración

Cuando la distancia sea distinta de 0 vamos a realizar los cálculos con objeto de obtener todos los datos necesarios para nuestra aplicación.

Por otra parte, observemos el fragmento de código contenido en el **código fuente 17** relativo a los atributos que almacenan el tiempo:

```
//Calculamos el tiempo entre los avisos sonoros
tiempoEntreNotificaciones += tiempo;
tiempoEntreNotificacionesMarcha += tiempo;
```

Código Fuente 17. Atributos de tiempo en onLocationChanged

Estos dos atributos se encargan de gestionar el tiempo que transcurre entre las notificaciones sonoras, el segundo de ellos almacena el tiempo que ha pasado entre los sonidos asociados a la marcha que deberíamos engranar en la caja de cambios y el primero se encarga del resto. Se han establecido estos márgenes entre notificaciones sonoras para no saturar al usuario con instrucciones. Dos métodos diferentes se encargan de reproducir los sonidos adecuados teniendo en cuenta estos márgenes establecidos.

Sigamos avanzando en nuestro pseudocódigo mediante el **código fuente 18**:

```
float velocidadActual = 0;

if (loc.hasSpeed()) {

    velocidadActual = loc.getSpeed();
    validarVelocidadDistancia(velocidadActual);

} else {
    velocidadActual = distancia / tiempo;
}
```

Código Fuente 18. Cálculos de velocidad en el método onLocationChanged()

En esta parte del algoritmo vamos a realizar los cálculos necesarios para obtener la velocidad a la que se mueve el dispositivo. Para empezar, creamos una variable auxiliar que utilizaremos para los cálculos. El método ***Location.hasSpeed()*** del API de Android devuelve ***"true"*** si el objeto *Location* contiene el valor de la velocidad y ***"false"*** en caso contrario. En el caso de que devuelva "false" calcularemos nosotros mismos la velocidad, ya que sabemos que:

velocidad = distancia / tiempo = m/s

Por el contrario, si el método devuelve "true", obtendremos la velocidad del propio objeto Location (en este caso llamado "**loc"**) ya que sabemos, gracias a la documentación del API proporcionada por Android, que el método ***Location.getSpeed*()** es más preciso que el método ***Location.distanteTo(Location)*** utilizado antes para obtener la distancia. Por esa razón se ha desarrollado un método llamado ***validarVelocidadDistancia*()**, este se encarga de comprobar la diferencia entre la velocidad devuelta por el método ***Location.getSpeed()*** y la calculada por nosotros gracias a la formula anterior, en el caso de que la diferencia sea importante (mayor de un 10 % por encima o por debajo) se descarta la distancia obtenida con el método ***Location.distanteTo(Location)*** y se calcula de nuevo la distancia mediante la siguiente fórmula:

*distancia = velocidad * tiempo = m*

En el **código fuente 19** vemos el pseudocódigo de dicho método:

```
void validarVelocidadDistancia(float velocidadActual) {

    float velocidadCalculada = distancia / tiempo;

    if((velocidadCalculada > velocidadActual * (float)1.1) ||
    (velocidadCalculada < velocidadActual * (float)0.9)) {

        distancia = velocidadActual * tiempo;
    }
 }
```

Código Fuente 19. pseudocódigo del método validarVelocidadDistancia()

El hecho de realizar estas comprobaciones cada vez que recibimos una coordenada nos otorga cierta latencia en la aplicación pero, a cambio, obtenemos una precisión en los cálculos notablemente mejor, por lo tanto, merece la pena llevar a cabo esta comprobación.

Ya hemos descrito el proceso que nos permite calcular la velocidad y la distancia, por último, nos queda mostrar cómo se realiza el cálculo de la aceleración. Dicho cálculo se realiza en el siguiente fragmento de código, mostrado en el **código fuente 20**, extraído del pseudocódigo mostrado anteriormente en el **código fuente 16**:

```
if(!segundaCoordenada) {

    aceleracion = (velocidadActual - velocidad) / tiempo;

    (…)
}
```

Código Fuente 20. Cálculo de la aceleración en el método onLocationChanged()

Como se puede comprobar en el **código fuente 20**, sólo calculamos la aceleración a partir de la tercera coordenada recibida, la aceleración se obtiene mediante un algoritmo sencillo:

aceleración = (velocidad final – velocidad inicial) / tiempo = m/s^2

Se ha omitido el fragmento del pseudocódigo que realiza los cálculos necesarios para mostrar los datos del viaje en la pantalla resumen. Se trata de cálculos sencillos que permiten obtener la aceleración máxima, la deceleración máxima, aceleración media, etc.

Siguiendo con el pseudocódigo del método ***onLocationChanged*()**, queda por mostrar la última parte que completa el algoritmo de dicho código, en el **código fuente 21** encontramos el pseudocódigo del fragmento que falta:

```
void onLocationChanged(Location loc) {

    (…)

    if(!primeraCoordenada) {

        (…)

    } else {
       primeraCoordenada = false;
    }

    //Comprobar si podemos apagar motor
    informarApagarMotor();

    locationAnterior = loc;

    imprimeAceleracion();
    textViewVelocidad.setText
        (Integer.toString((int)(velocidad * (float)3.6)));

    //Para la pendiente
    calcularPendiente(loc);

    muestraColores();
    calcularMarchaOptima();
}
```

Código fuente 21. Última parte pseudocódigo del método onLocationChanged()

Una vez descrito el pseudocódigo incluido en el bloque "**if**", vamos a continuar con el resto del pseudocódigo, lo primero que encontramos es una llamada a un método ajeno a este, se trata de la llamada al método ***informarApagarMotor*()**.

El método ***informarApagarMotor*()**, que también se encuentra en la clase **MyLocationListener**, comprueba si el dispositivo lleva detenido 30 segundos, en ese caso, muestra en pantalla un mensaje indicando al usuario que en paradas prolongadas (mayores de 60 segundos) es recomendable apagar el motor, por supuesto, este mensaje va acompañado del correspondiente aviso sonoro.

El algoritmo de dicho método es muy sencillo, en el caso de que la velocidad sea 0 (el valor del atributo llamado "**velocidad"**), suma al atributo "**tiempoParada"** el valor del atributo "**tiempo"**, que almacena el tiempo entre las dos últimas coordenadas recibidas. De esta forma, cuando el valor del atributo "**tiempoParada"** alcanza o supera los 30 segundos se lanzan los mensajes descritos en el apartado anterior. En el caso de

que la velocidad sea distinta de 0, el valor del atributo "**tiempoParada"** se establece a 0 y no se muestra mensaje alguno.

Siguiendo con el pseudocódigo del método ***onLocationChanged*()**, en la siguiente línea, después de la llamada al método anterior, se actualiza el atributo "**locationAnterior**" con el objeto **Location** recibido como argumento del método. Así, este último objeto **Location** recibido nos servirá para hacer los cálculos oportunos cuando llegue el próximo objeto de este tipo (la próxima coordenada).

A continuación de la sentencia descrita encontramos otra llamada a un método, en este caso, la llamada es al método ***imprimeAceleracion*()**. Este método utiliza el valor del atributo "**aceleracion**" para mostrar el valor de la aceleración actual en la etiqueta "**textViewAceleracion**".

En la siguiente sentencia, se muestra en la etiqueta "**textViewVelocidad"** el valor actual de la velocidad, almacenado en el atributo llamado "**velocidad"**. Es necesaria una conversión en las unidades ya que en la etiqueta se muestra la velocidad en *Km/h* y en el código trabajamos con *m/s* como unidades.

En la siguiente línea encontramos la llamada al método ***calcularPendiente*()**, este método se encarga de calcular la pendiente de la vía por la que circula el vehículo, también se encarga de cuantificar como afecta al consumo del vehículo el hecho de circular por una pendiente, por último, se encarga de adaptar los márgenes establecidos para la aceleración y para la velocidad de acuerdo a dicha pendiente a partir del resultado obtenido respecto al consumo relativo a la pendiente. Este método contiene un algoritmo complejo por lo que se mostrará posteriormente en un apartado específico.

Por otro lado, el método ***muestraColores*()** compara los datos obtenidos de aceleración y velocidad con los valores de los márgenes establecidos y modifica los colores de la pantalla en concordancia con los resultados. Por ejemplo, en el caso de que el usuario supere los límites máximos de velocidad o aceleración, se mostrará el color rojo como color de fondo y en las etiquetas correspondientes, en el caso de que el usuario circule por debajo de dichos límites, se mostrará el color verde o amarillo en esos mismos lugares. Se han establecido 5 colores diferentes formando una escala progresiva que varía desde el rojo hasta el verde pasando por el color amarillo (fijado para el límite recomendable de velocidad y aceleración).

En la última línea del pseudocódigo encontramos la llamada al método ***calcularMarchaOptima*()**. Dicho método se encarga de mostrar en pantalla y mediante avisos sonoros la marcha de la caja de cambios adecuada para la velocidad a la que circula el usuario. Los límites para cada marcha son los indicados por las 10 claves de la conducción eficiente según el IDAE. En el caso de los avisos sonoros, sólo se reproducirá el sonido avisando de la marcha adecuada si es necesario un cambio en la marcha engranada y si han pasado más de 5 segundos desde el último aviso.

3.2.4.6 Algoritmo de cálculo de pendientes

En este apartado vamos a centrarnos en el algoritmo que se encarga de gestionar los datos de altura, respecto al nivel del mar, obteniendo a partir de ellos la pendiente de la vía por la que circula el vehículo. Mediante la pendiente se hace una estimación del incremento o decremento del gasto de combustible que supone circular en esas condiciones frente a hacerlo en una vía completamente llana. El objetivo final de este algoritmo es adaptar los límites establecidos por el usuario, sobre aceleración y velocidad, ajustándolos a la pendiente de la vía por la que circulamos.

Para poder determinar cómo afecta la pendiente de la vía al consumo, el primer paso es determinar dicha pendiente. A través del GPS podemos obtener la pendiente de la vía por la que circulamos mediante las lecturas de altitud sobre el nivel del mar que contienen los objetos **Location**.

Cabe destacar que la precisión en la posición vertical (altitud) es del orden de tres veces peor que la precisión horizontal, por ejemplo, si en horizontal estamos trabajando con unos valores que poseen una precisión de 10 metros, en vertical, sin en cambio, la precisión será de unos 30 metros. Esta precisión es incompatible con nuestra aplicación, teniendo en cuenta que la pendiente normal de una vía suele oscilar entre el 0 y el 20% (una pendiente del 20 % indica que cada 100 metros de vía hay una diferencia en la altura de 20 metros), un error de 30 metros produciría unos datos completamente alejados de la realidad.

Para solucionar o, por lo menos, reducir significativamente este error producido por la precisión de los datos se ha diseñado un algoritmo que utiliza varias medidas de altitud para determinar la altitud real. Esta solución consiste en tomar las 5 últimas muestras recibidas, que estarán separadas entre sí 1 segundo si la visibilidad con los satélites es buena. A partir de estas 5 muestras se eliminan la mayor y la menor (de esta manera reducimos la dispersión de los datos), con las tres muestras restantes se realiza la media y ese es el valor de altitud que vamos a tomar. Vamos ahora a realizar un ejemplo de este algoritmo para el cálculo de alturas:

Estos podrían ser los últimos 5 valores de altitud asociados a los objetos **Location** recibidos: 705, 710, 707, 705 y 693 metros.

1º Descartamos el mayor y el menor: 710 y 693 metros.

2º Hacemos la media de los valores restantes: (705 + 707 + 705) / 3 = 705,66 m.

Si no eliminásemos el menor y el mayor la media vendría desplazada por el valor que está más alejado del resto, en este caso 693, por lo que el resultado sería 704 m.

Para calcular la pendiente necesitamos dos valores de altitud y la distancia recorrida entre esos dos valores. De este modo, calcularemos nuestra pendiente a partir de diez muestras, 5 para el primer valor de altitud y otras 5 para el segundo, sumando la

distancia total recorrida durante esas muestras. El porcentaje de la pendiente lo obtendremos entonces mediante la siguiente relación:

*Pendiente (%) = (Altitud final – Altitud inicial) * 100 / distancia recorrida*

A modo de ejemplo, vamos a tomar los datos del ejemplo anterior suponiendo que la distancia recorrida entre todas las muestras, sumando las distancias parciales de las 5 muestras, es de 152,6 m.

Siguiendo con el ejemplo, necesitamos otras 5 muestras para calcular la pendiente, estas pueden ser: 721,723, 740, 718 y 719 m. Aplicando el algoritmo para el cálculo de la altitud, la altitud resultante sería 721 m. Suponiendo para estas muestras una distancia total recorrida de 154,2 m. Podríamos calcular la pendiente como:

Pendiente (%) = (721 – 705.66) * 100 / (152,6 +154,2) = 5%

El resultado del ejemplo anterior es una pendiente ascendente del 5%.

El método encargado de realizar esta tarea pertenece a la clase **MyLocationListener** y se llama ***calcularPendiente*()**. Este es su pseudocódigo, **código fuente 22**:

```
void calcularPendiente(Location loc) {

   if(distancia != (float)0) {

      vector_altitudes.add((float)loc.getAltitude());
      distanciaCalcularPendienteNueva += distancia;

      if(vector_altitudes.size() >= 5) {

         float valor_minimo = vector_altitudes.get(0);
         int pos_minimo = 0;
         float valor_maximo = vector_altitudes.get(0);
         int pos_maximo = 0;

         for (int i=1; i<5; i++) {

            float valorAux = vector_altitudes.get(i);

            if(valorAux < valor_minimo) {

               valor_minimo = valorAux;
               pos_minimo = i;

            } else if (valorAux > valor_maximo) {

               valor_maximo = valorAux;
               pos_maximo = i;

            }
         }

         float sumaAlturas = 0;
```

```
            //Vamos a contar los datos que sumamos para obtener la
            media posteriormente
            int contadorAltMedia = 0;

            for(int i=0; i<5; i++) {

                if((i == pos_maximo) || (i == pos_minimo)) {
                    continue;
                }

                sumaAlturas += vector_altitudes.get(i);
                contadorAltMedia++;
            }

            float altitudNueva =
            sumaAlturas / (float)contadorAltMedia;

            float distanciaTotalAltitud =
        distanciaCalcularPendienteNueva +
        distanciaCalcularPendiente;

            int pendientePorcentaje =
calcularGradoPendiente(altitudNueva, distanciaTotalAltitud);

            textPendiente.setText
            (Integer.toString(pendientePorcentaje));

            //Mostramos al usuario la diferencia en el consumo debida
            a la pendiente
            int porcentajeVariacionConsumo = 0;
            if(factorPendiente >= 1) {

                porcentajeVariacionConsumo =
      (int)((factorPendiente - 1) * 100) * (-1);

            } else {

                porcentajeVariacionConsumo =
      (int)((1 - factorPendiente) * 100);
            }

            textVariacionConsumo.setText
            (Integer.toString(porcentajeVariacionConsumo));

            (…)

            vector_altitudes.removeAllElements();

            distanciaCalcularPendiente =
            distanciaCalcularPendienteNueva;

            distanciaCalcularPendienteNueva = 0;
        }
    }
}
```

Código Fuente 22. Pseudocódigo del método calcularPendiente()

El método recibe como argumento el último objeto **Location** enviado por el GPS. La primera comprobación a realizar es que la distancia sea distinta de 0, no tiene sentido

calcular la pendiente si el vehículo está parado ya que los datos de altitud no varían en ese caso. En la etiqueta que muestra el valor de la pendiente en la interfaz de usuario se seguirá mostrando, por tanto, el último valor calculado de la pendiente.

En la siguiente línea del **código fuente 22** encontramos el atributo "**vector_altitudes**", este atributo de tipo **Vector<Float>** almacena una colección de datos tipo **float** referentes a la altitud recibida en cada objeto **Location**. En este vector almacenaremos los 5 datos de altitud necesarios para calcular la altitud final. Es por eso que en esta primera línea guardamos el valor de altitud almacenado en el objeto "**loc**" recibido dentro de este vector.

Una vez almacenado el dato de altitud, el siguiente paso es almacenar la distancia recorrida entre esta coordenada y la anterior. Para ello, hacemos uso del atributo "**distanciaCalcularPendienteNueva**", también de tipo **float**. A este atributo le sumamos el valor del atributo "**distancia"** para cada una de las cinco coordenadas necesarias, así obtendremos la distancia total.

Cuando en el objeto "**vector_altitudes"** haya 5 datos guardados podremos hacer las operaciones necesarias para obtener la altitud media y la pendiente. En las siguientes líneas encontramos un bucle **for** cuyo objetivo es recorrer el vector en busca del dato de altitud más alto y el más bajo. En la variable "**pos_minimo"** almacenaremos el índice del lugar que ocupa el dato mínimo de altitud dentro del vector y en la variable "**pos_maximo"** guardaremos la posición del máximo.

A continuación, vamos a obtener la altitud media, para ello, vamos a sumar los valores de altitud guardados en el vector, discriminando el máximo y el mínimo. Una vez obtenida la suma de las altitudes dividimos entre el número de altitudes sumadas y así obtenemos la media. Este valor medio obtenido lo almacenamos en la variable llamada "**altitudNueva**".

En la variable tipo **float** llamada "**distanciaTotalAltitud**" guardamos la distancia total recorrida durante los diez últimos objetos **Location** obtenidos, para poder calcular así la pendiente. Para ello sumamos el valor del atributo "**distanciaCalcularPendienteNueva**", que contiene la distancia de las 5 últimas coordenadas, más el valor del atributo "**distanciaCalcularPendiente**", que guarda la distancia total de las 5 coordenadas anteriores.

Vamos ahora a obtener la pendiente, en porcentaje, de la vía y la forma en la que esta afecta al consumo. Para ello, vamos a hacer una llamada al método encargado de calcular estos factores, descrita en el **código fuente 23**:

```
    int pendientePorcentaje = calcularGradoPendiente(altitudNueva,
distanciaTotalAltitud);
```

Código Fuente 23. Llamada al método calcularGradoPendiente()

El método se llama ***calcularGradoPendiente*()** y se encuentra en esta misma clase. Este método recibe como argumentos la última altitud calculada y la distancia total recorrida durante las últimas 10 muestras.

Este método es muy sencillo. Primero comprueba que el valor del atributo "**altitud**" sea distinto de cero, lo que indica que ya se ha calculado previamente una altitud, ya que no podemos calcular la pendiente con una sola altitud. Posteriormente, el método calcula el porcentaje de la pendiente mediante la relación descrita anteriormente. Una vez obtenida la pendiente, este método realiza una llamada al método ***calcularFactorPendiente*()** encargado de cuantificar el valor de la relación entre la pendiente de la vía y el consumo del vehículo (veremos posteriormente en detalle el algoritmo contenido en este método), el valor calculado se guardará en el atributo llamado "**factorPendiente**". Por último, el método ***calcularGradoPendiente*()** actualiza el valor del atributo "**altitud**" con el valor recibido como parámetro ("**altitudNueva**") y devuelve el valor de la pendiente calculada.

Continuando con el pseudocódigo del método ***calcularPendiente*()**, una vez calculado el valor de la pendiente de la vía, nos encontramos el cálculo de otro dato que será mostrado al usuario en la pantalla, asociado al valor de la pendiente. El siguiente fragmento de código se encarga de calcular el valor de la variable que hemos llamado "**porcentajeVariacionConsumo**", esta variable almacena el incremento o decremento del consumo en función de la pendiente calculada, se basa en el valor de "**factorPendiente**" para extraer de él el porcentaje de variación del consumo debido a la pendiente, este valor será mostrado al usuario en la interfaz como información adicional interesante para el ahorro de combustible.

Por ejemplo, en el caso de que el atributo "**factorPendiente**" tenga el valor 1'12, la variable "**porcentajeVariacionConsumo**" tomará el valor -12%, esto indicará al usuario que el consumo es ahora un 12% menos gracias a la pendiente descendente. En el caso de que el vehículo circule por una pendiente ascendente, si el valor de "**factorPendiente**" es, por ejemplo, 0'97, el valor de "**porcentajeVariaciónConsumo**" sería 3%, indicando entonces dicho incremento en el consumo.

Si seguimos avanzando en el pseudocódigo del método ***calcularPendiente*()** encontramos ahora un fragmento de código suprimido, en este fragmento se hacen comprobaciones para pintar de un color u otro la etiqueta que muestra la pendiente en la interfaz de usuario y se realizan operaciones para poder calcular la pendiente media, parámetro que será mostrado en la pantalla resumen.

En las últimas líneas del pseudocódigo se actualizan los valores de algunos atributos para que puedan ser utilizados de nuevo con el objeto de volver a calcular las futuras pendientes de la vía que nos encontremos durante el viaje.

3.2.4.7 Algoritmo de cálculo del factor pendiente

Una vez concluido el pseudocódigo del método ***calcularPendiente*()** vamos ahora a volver sobre el método ***calcularFactorPendiente*()**, mencionado en el apartado anterior. Este método se encarga de estimar cuanto afecta al consumo del vehículo la pendiente actual por la que circulamos y ajusta los límites de la conducción en función de ese parámetro, esta es su llamada (**código fuente 24**) dentro del método ***calcularGradoPendiente*()**:

```
calcularFactorPendiente(pendiente);
```

Código Fuente 24. Llamada al método calcularFactorPendiente()

Se trata de un método relativamente sencillo, no obstante, se han tomado algunas decisiones para su diseño referentes a la adaptabilidad de los límites de aceleración y velocidad en función de la pendiente de la vía. A continuación, describiremos esas decisiones y mostraremos algunos ejemplos del funcionamiento del algoritmo desarrollado.

Como sabemos, cuando un vehículo se mueve actúan sobre él varias fuerzas en diferentes direcciones. Algunas de estas fuerzas se oponen al movimiento del vehículo, como el rozamiento de rodadura (de las ruedas con la superficie de la vía), el rozamiento de las diferentes piezas del motor entre sí, la resistencia del aire a ser desplazado, etc. La energía necesaria para vencer esas fuerzas contrarias al movimiento procede del motor del vehículo, se produce quemando combustible.

Las fuerzas citadas en el apartado anterior varían en función de diversos factores, siendo la velocidad el factor más importante, sin en cambio, la pendiente de la vía por la que circulemos no influye sobre estas fuerzas. Es por ello que en esta ocasión no vamos a tenerlas en cuenta.

Al margen de las fuerzas de rozamiento existe una fuerza que mantiene el vehículo pegado al suelo, es el peso. El peso tiene que ver con la fuerza con que la Tierra atrae a un cuerpo que posee una cierta masa (**m**). Sabemos que el peso es el resultado de multiplicar la masa del cuerpo por la aceleración de la gravedad (**g**) del planeta en el que se encuentre el cuerpo, en este caso la tierra (en la Tierra el valor de la aceleración de la gravedad se puede aproximar como 10 m/s^2), esta es la relación que nos permite calcular dicha fuerza:

*Peso = masa * gravedad (g) = kg * m/s^2 = Newtons (N)*

El peso actúa sobre el cuerpo en dirección hacia el centro de masas de la Tierra. Esto convierte esta fuerza en relevante para nosotros en este apartado. Cuando el vehículo circula por una pendiente, el peso ejerce una fuerza que intenta desplazar el vehículo hacia el punto más bajo de esta. Esto afecta directamente al consumo de combustible de nuestro vehículo ya que, en el caso de circular por una pendiente ascendente, el peso va a ejercer una fuerza en la dirección opuesta al desplazamiento

que tendrá que vencer el motor para que el vehículo continúe avanzando en la dirección deseada, por el contrario, cuando circulamos por una pendiente descendente, el peso ejerce dicha fuerza en la dirección del movimiento por lo que ayuda al motor en la tarea de desplazar el vehículo.

Como vimos en el apartado sobre conducción eficiente (apartado 2.3), de la energía que se produce al quemar un litro de combustible, la mayor parte se pierde por ineficiencias del propio motor y sólo se transmite a las ruedas del vehículo aproximadamente un 15% de esa energía producida. Además, la mayor parte de esa energía se pierde por rozamiento, ya sea el producido por el aire que rodea al vehículo o por el rozamiento de rodadura (el producido al arrastrar las ruedas por la superficie de la vía). Podemos deducir, por tanto, que el movimiento del vehículo será el resultado de aquella energía que no se ha perdido a causa de todos los demás factores.

Hablando en términos de energía, sabemos que la energía mecánica (E_{mec}) se debe a la posición y al movimiento de un cuerpo que está sometido a fuerzas. Esta energía es el resultado de sumar la energía cinética del cuerpo (E_c) más su energía potencial (E_p). Siendo un poco más específicos, la energía mecánica expresa la capacidad que posee un cuerpo con cierta masa de realizar un trabajo. En un entorno sin rozamiento la energía mecánica se conserva, en este caso, el rozamiento y la energía proporcionada por el motor hacen que el valor de esta energía no sea constante. Esta es su ecuación:

$$E_{mec} = E_c + E_p = Julios\ (J)$$

Por su parte, la energía cinética es la energía asociada al movimiento, depende de la masa y de la velocidad del cuerpo según la relación:

$$E_c = ½ * masa * velocidad^2$$

Por otro lado, la energía potencial depende de la posición del cuerpo. Más concretamente, se dice que un cuerpo de masa **m** situado a una altura **h** está sometido a una aceleración gravitatoria y por lo tanto, posee energía. Esta energía viene determinada por la siguiente relación:

$$E_p = m * g * h = peso * altura\ (h)$$

Haciendo uso de estas relaciones, podemos calcular la energía mecánica del vehículo en su desplazamiento. Asumiremos que esta energía mecánica es la energía efectiva que mueve el vehículo, es decir, las variaciones de la energía mecánica marcarán la diferencia de consumo del vehículo.

Para nuestra aplicación, queremos cuantificar la diferencia sobre el gasto de combustible en función de la pendiente. Para ello, es fácil observar que si suponemos una velocidad constante, la diferencia entre la energía mecánica de un vehículo que circula por una vía sin pendiente y el mismo vehículo que lo hace por un tramo con pendiente dependerá únicamente de la diferencia en la energía potencial entre dos instantes de tiempo.

A la hora de calcular la energía potencial de un cuerpo hay que tener en cuenta que es irrelevante donde coloquemos el origen de coordenadas ya que lo realmente indicativo es la diferencia de esa energía potencial entre dos puntos. Por tanto, en un coche que circula a velocidad constante entre dos puntos (A y B) podemos calcular su energía mecánica mediante la siguiente relación:

$$E_{mec} = E_c + \Delta E_p = ½ * m * v^2 + m * g * (h_B - h_A)$$

En la relación mostrada, ΔE_p indica que para calcular la energía potencial hemos tomado como origen de coordenadas el punto A.

En el caso de que el coche circule por un tramo sin pendiente, la altura en el punto B sería la misma que en el punto A, por lo tanto, la variación de su energía potencial es 0, es decir, su energía mecánica solo depende de su energía cinética.

Por el contrario, si el vehículo circulara a la misma velocidad que antes por un tramo en pendiente, la variación de su energía potencial sería distinta de 0. Es decir, además de la energía necesaria para mantener la velocidad, necesitaría una cantidad extra de energía para ascender por la pendiente sin perder velocidad, o bien, en el caso de que la variación de su energía potencial fuese negativa, necesitaría menos energía mecánica para mantener esa velocidad, esto ocurriría en una pendiente descendente.

Para cuantificar la diferencia de consumo entre los diferentes casos (pendiente de subida, de bajada o tramo llano) hemos creado una variable que se llama "**relacionEnergiaPendiente**". A la hora de calcular este valor hay que hacer una consideración, cuando calculamos la diferencia en la energía potencial entre dos puntos distintos obtenemos el valor total de esa diferencia, en nuestro caso eso no significa que para llevar el vehículo desde un punto a otro el motor deba producir esa diferencia de energía instantáneamente. En realidad, lo que ocurrirá es que el motor irá produciendo la energía progresivamente, mientras avanza por la vía, hasta que alcance el punto final. Esto significa que en el caso que nos ocupa no nos interesa tanto la diferencia en la energía potencial entre dos puntos como el incremento o decremento de la energía potencial en un instante de tiempo.

Debido a lo descrito en el párrafo anterior, vamos a calcular esa diferencia de la energía potencial en un periodo corto de tiempo, más concretamente, en un periodo de 1 segundo. Podemos calcular este valor a partir de la velocidad y la pendiente. Multiplicando la velocidad por el tiempo y por la pendiente obtenemos la diferencia en la altura para un periodo de 1 segundo, de esta forma conseguimos obtener una relación entre la pendiente y el consumo mucho más cercana a la realidad que usando la diferencia de la altura entre dos puntos alejados entre sí. En nuestro caso, ya que obtenemos la pendiente de la vía a partir de 10 muestras, siempre que el vehículo circule a una velocidad alta estos dos puntos pueden estar muy alejados entre sí, la consecuencia sería que la diferencia de la energía potencial entre esos dos puntos podría ser muy grande. Por el contrario, usando el método descrito, obtenemos la diferencia de

la energía potencial por instante de tiempo. Para calcular este valor se ha establecido la siguiente relación:

relacionEnergiaPendiente $= E_{mec}$ *(Pendiente)* $/ E_{mec}$ *(Llano)* $= (E_c + \Delta E_p) / E_c$

Como se puede observar, se trata de una relación entre la energía mecánica del vehículo que se mueve por una pendiente y la energía mecánica del mismo vehículo si este circulara por un tramo llano a la misma velocidad. Aplicando la fórmula elaborada para obtener la diferencia en altura entre dos puntos (h_B y h_A) separados una distancia igual a la velocidad por el tiempo (1 segundo) obtenemos la siguiente relación:

relacionEnergiaPendiente $= (½*m*v^2 + m * g * (h_B - h_A)) / ½*m*v^2 =$

$= (½*m*v^2 + m * g * (velocidad * tiempo\ (1s) * (pendiente/100))) / ½*m*v^2$

Ya que:

$h_B - h_A = velocidad * tiempo(1s) * (pendiente/100)$

La razón de dividir el valor de la pendiente entre 100 es que esta ha sido calculada en porcentaje (%) por lo que para poder usarla en la relación es necesaria esta operación previa.

Para poder aplicar el resultado obtenido de la relación anterior sobre los límites de aceleración y velocidad establecidos por el usuario es necesaria una pequeña conversión. De la ecuación anterior extraemos que cuando se trate de una pendiente ascendente esta relación va a dar como resultado un número mayor que 1, puesto que el numerador es mayor que el denominador. Por el contrario, en una pendiente descendente (negativa), h_B será menor que h_A de modo que el numerador será más pequeño que el denominador y, por consiguiente, el resultado será un número menor que 1.

Es necesario para nuestra aplicación que los márgenes sean más restrictivos en una pendiente de subida que en una pendiente de bajada, debido a esto no podemos multiplicar nuestros márgenes directamente por la relación anterior, para poder hacerlo hemos creado un atributo llamado "**factorPendiente**" que calculamos a partir del resultado anterior de la siguiente manera:

- Si el resultado es mayor o igual que 1:

  factorPendiente = 2 – **relacionEnergiaPendiente**

- Si el resultado es menor que 1:

  factorPendiente = (1 - **relacionEnergiaPendiente**) + 1

Este "**factorPendiente**" se puede multiplicar directamente por los límites de aceleración y velocidad y así obtenemos dichos límites ajustados a la pendiente de la vía. En el caso de una pendiente ascendente, el "**factorPendiente**" será un número

menor que uno, esto producirá que los límites de la conducción sean más restrictivos. Por el contrario, si la pendiente es descendente, este número será mayor que 1 y producirá límites en la aceleración y la velocidad más permisivos, indicando de esta forma un menor grado de consumo.

Para calcular el "**factorPendiente**" se han establecido algunas condiciones:

- La masa del vehículo se ha establecido en 120 Kg (1200 Kg de peso en la Tierra) como valor medio de la masa de un vehículo turismo.
- La aceleración de la gravedad (g) se aproxima como 10 m/s^2.
- Como se ha comentado, la diferencia en la energía potencial producida por la pendiente se calculará para periodos de 1 segundo.

A continuación, mostraremos algunos ejemplos del cálculo del valor de "**relacionEnergiaPendiente**" con el objetivo de ilustrar un poco mejor como funciona esta relación en diferentes condiciones. Compararemos lo que ocurre a tres velocidades diferentes con cuatro pendientes distintas, donde m = 120 Kg, g = 10 m/s^2 y t = 1 segundo:

La energía cinética en cada uno de los casos es:

Ec (20 Km/h = 5,55 m/s) = 1848,15 J

Ec (50 Km/h = 13,5 m/s) = 10.935 J

Ec (100 Km/h = 27,7 m/s) = 46.037,4 J

La relación entre las energías mecánicas ("**relacionEnergiaPendiente**") obtenida a 20 Km/h para cada una de las pendientes:

(20Km/h y pendiente 2%) = (1.848,15 + 120*10 *5,55 * (2/100)) / 1.848,15 = 1,072

(20Km/h y pendiente 5%) = 1,18

(20Km/h y pendiente 10%) = 1,36

(20Km/h y pendiente 20%) = 1,72

(20Km/h y pendiente -5%) = (1.848,15 + 120*10*5,55*(-5/100)) / 1.848,15 = 0,82

La relación entre las energías mecánicas ("**relacionEnergiaPendiente**") obtenida a 50 Km/h para cada una de las pendientes:

(50Km/h y pendiente 2%) = (10.935 + 120 * 10 * 13,5 * (2/100)) / 10.935 = 1,03

(50Km/h y pendiente 5%) = 1,074

(50Km/h y pendiente 10%) = 1,148

(50Km/h y pendiente 20%) = 1,296

(50Km/h y pendiente -5%) = (10.935 + 120 * 10 * 13,5 * (-5/100)) / 10.935 = 0,926

La relación entre las energías mecánicas ("**relacionEnergiaPendiente**") obtenida a 100 Km/h para cada una de las pendientes:

(100Km/h y pendiente 2%) = (46.037,4 + 120*10*27,7*(2/100)) / 46.037,4 = 1,014

(100Km/h y pendiente 5%) = 1,036

(100Km/h y pendiente 10%) = 1,072

(100Km/h y pendiente 20%) = 1,144

(100Km/h y pendiente -5%) = (46.037,4+120*10*27,7*(-5/100)) /46.037,4 = 0,964

De los resultados obtenidos mediante los ejemplos podemos extraer algunas conclusiones:

- A mayor pendiente ascendente, mayor es la energía necesaria para subirla, mas aumenta el consumo.
- A mayor pendiente descendente, mayor es la energía que contribuye a mantener la velocidad, disminuyendo por tanto la necesidad del motor de generar esa energía.
- A velocidades bajas la energía necesaria para subir una pendiente es más representativa que la necesaria para mantener la velocidad por lo que se aumenta proporcionalmente mucho más el consumo en esa situación respecto al consumo medio. O se disminuye en el caso de una pendiente descendente.
- A velocidades altas la energía cinética del vehículo es muy superior a la variación de la energía potencial por lo que la pendiente influye de forma menos representativa sobre el consumo real del vehículo.

A la vista de los resultados obtenidos, observamos que las conclusiones extraídas se acercan mucho a la realidad de un vehículo que circula por una pendiente. Parece, por tanto, una buena aproximación a la relación real entre la pendiente y el consumo del vehículo.

Hay que tener en cuenta que no podemos aplicar directamente esta relación sobre los límites de aceleración y velocidad, para ello, tenemos que calcular el "**factorPendiente**" de la manera en la que se indicó anteriormente. Por ejemplo:

factorPendiente (20Km/h y pendiente -5%) = (1 - 0,82) + 1 = 1,18

factorPendiente (100Km/h y pendiente 10%) = 2 - 1,072 = 0,928

En el primer caso, los límites de la aceleración y la velocidad serán cada uno un 18% mayor. En el segundo caso, cada uno de estos límites será un 7,2% más restrictivo.

Una vez justificadas las decisiones tomadas al respecto, podemos proseguir con el pseudocódigo del método ***calcularFactorPendiente*()** de la clase **MyLocationListener**,

cuya llamada dentro del método ***calcularGradoPendiente*()** vimos en el **código fuente 24.** El pseudocódigo de dicho método se muestra en el **código fuente 25**:

```
   private void calcularFactorPendiente(float pendiente) {

       if(pendiente != 0) {

           //masa coche = 120 kg, en peso 1200 kg, la aceleración de la
           gravedad 10 m/s²
           float energiaCineticaFinal =
(float)60 * (float)Math.pow((double)velocidad, (double)2);

           //Calculamos la variación de la energía potencial para la
           velocidad y pendiente dada por cada segundo
           float variacionEnergiaPotencialPendiente =
masaPorGravedad * velocidad * (pendiente/100);

           float relacionEnergiaPendiente = (energiaCineticaFinal +
variacionEnergiaPotencialPendiente) / energiaCineticaFinal;

           //Si la pendiente es ascendente reduzco los márgenes, en
           caso contrario los aumento
           if(relacionEnergiaPendiente >= (float)1) {

               factorPendiente = (float)2 - relacionEnergiaPendiente;

           } else {

               factorPendiente = ((float)1 - relacionEnergiaPendiente) +
(float)1;

           }

       } else {

           factorPendiente = 1;

       }
   }
```

Código fuente 25. Pseudocódigo del método calcularFactorPendiente()

En el pseudocódigo se realizan las mismas operaciones que hemos visto anteriormente. Se ha añadido una comprobación al principio del método que impide realizar el cálculo del "**factorPendiente**" en el caso de que no exista pendiente en este tramo. Si existe pendiente, empezamos calculando la energía cinética del vehículo. Después la variación de la energía potencial en función de la pendiente que hemos calculado. Existe un atributo que se llama "**masaPorGravedad**"**,** este almacena el valor del peso del vehículo (m * g = 120 * 10 = 1200 N), gracias a este atributo sólo realizamos esta operación una vez.

A partir de los valores obtenidos, podemos calcular el valor de "**relacionEnergiaPendiente**" utilizando el método descrito con anterioridad. Con este dato calculamos el "**factorPendiente"**. Para aplicar el "**factorPendiente"** en los límites sobre aceleración y velocidad no vamos a modificar estos directamente, en su lugar,

emplearemos este valor sobre los límites cuando realicemos el diagnóstico de la conducción con los datos calculados (aceleración y velocidad) de la siguiente manera:

```
public int calculaGradoConsumoVelocidad() {

    int grado = 0;

    if (velocidad >= velocidadConsMax * factorPendiente) {
        //Rojo
        grado = 5;

    } else if (velocidad >= velocidadIntermediaConsMax *
    factorPendiente) {
        //Rojo&Amarillo
        grado = 4;

    } else if (velocidad >= velocidadEco * factorPendiente) {
        //Amarillo
        grado = 3;

    } else if (velocidad >= velocidadIntermediaEco *
    factorPendiente) {
        //Amarillo&Verde
        grado = 2;

    } else {
        //Verde
        grado = 1;
    }

    return grado;
}
```

Código Fuente 26. Pseudocódigo del método calcularGradoConsumoVelocidad()

Como se puede observar en el **código fuente 26,** en función de la velocidad obtenida del GPS, se toma la decisión de cómo es la conducción respecto a los límites de velocidad establecidos, aplicamos el "**factorPendiente"** sobre dichos límites para no modificar el valor de estos ya que los usaremos como referencia durante la ejecución de la aplicación.

Se han establecido cuatro límites, estos proporcionan cinco estados, acerca de la velocidad del vehículo, para cada uno de estos estados se mostrarán unos mensajes y unos colores determinados. Para este cometido son necesarios cuatro atributos de clase:

- **velocidadConsMaxima**: es el límite máximo permitido para la velocidad establecido por el usuario, el valor de este atributo se obtiene del objeto **Intent** que envía la actividad "**AhorroConfiguracionPrevia**".

- **velocidadEco**: es el límite máximo recomendable establecido para realizar una conducción eficiente, el valor de este atributo se obtiene del objeto **Intent** que envía la actividad "**AhorroConfiguracionPrevia**".

- **velocidadIntermediaConsMax**: es el punto intermedio entre los dos límites anteriores.

- **velocidadIntermediaEco**: es la mitad de "**velocidadEco**".

Comparando el valor de la velocidad actual con estos límites podemos evaluar la conducción del usuario. El resultado de la evaluación se almacena en la variable "**grado**"**.** Cuando esta variable toma el valor 1 indica que la velocidad es más baja que la mitad del límite "**velocidadIntermediaEco**", eso quiere decir que respecto a la velocidad estamos en la zona de mayor ahorro de combustible. Un grado 5 indica que la velocidad actual del vehículo es superior o igual al límite máximo establecido (multiplicado por el factor pendiente), por consiguiente, nos encontraremos entonces en la zona en la que el gasto de combustible es máximo.

En el caso de la aceleración, el método que implementa esta funcionalidad se llama ***calcularGradoConsumoAceleración*()**. El algoritmo de ese método es el mismo que el método anterior, además utiliza también cuatro atributos gemelos a los de la velocidad, en este caso, el nombre de dichos atributos empieza por "**aceleracion**".

3.2.4.8 Intercambio de datos entre las actividades AhorroConfiguracionPrevia y AhorroCombustible

En este apartado vamos a describir los datos que intercambian las actividades entre sí. Este proceso se va a llevar a cabo a través de los objetos tipo **Intent**. En nuestra aplicación sólo se lanzan dos objetos **Intent**. La actividad **AhorroConfiguracionPrevia** lanza la actividad **AhorroCombustible** mediante un objeto **Intent**. El segundo de estos objetos **Intent** es utilizado por la actividad **AhorroCombustible** para inicializar la actividad **AhorroResumen**. Además de arrancar dichas actividades, los objetos **Intent** entregan a estas la información necesaria para comenzar con su tarea.

Vamos a comenzar describiendo como la actividad **AhorroConfiguracionPrevia** llama a la actividad **AhorroCombustible** mediante el objeto **Intent** correspondiente y los datos que este almacena. Este proceso comienza a través del siguiente método mostrado en el **código fuente 27**:

```
public void configurar(View view) {

   boolean aceleracionDefecto = false;
   boolean velocidadDefecto = false;

   Intent intent = new Intent(this, AhorroCombustible.class);

   aceleracionDefecto = obtenerAceleracionesUsuario(intent);
   velocidadDefecto = obtenerVelocidadesUsuario(intent);

   if(aceleracionDefecto || velocidadDefecto) {
```

```
            Dialog dialogo =
crearDialogoConfirmacion(aceleracionDefecto, velocidadDefecto,
intent);
            dialogo.show();

        } else {
            startActivity(intent);
        }
    }
```

Código Fuente 27. Método configurar() de la clase AhorroConfiguracionPrevia

Este método pertenece a la clase **AhorroConfiguracionPrevia**, es llamado cuando el usuario pulsa el botón "comenzar" de la interfaz. Como podemos observar en el **código fuente 27** este método crea un objeto tipo **Intent** llamado "***intent***" que será el encargado de llamar a la siguiente actividad.

El método ***obtenerAceleracionesUsuario*()** se encarga de recoger la información seleccionada por el usuario en los objetos SeekBar referentes a la aceleración e incluirla en el objeto **Intent** de la siguiente manera (**código fuente 28**):

```
intent.putExtra("ACELERACION_ECO", aceleracionEcoString);
intent.putExtra("ACELERACION_MAX", aceleracionConsMaxString);
```

Código Fuente 28. Introducción de datos de aceleración en el objeto Intent

El primer argumento del método ***putExtra*()** es la etiqueta que va a identificar a dicho dato dentro del objeto **Intent**. El segundo argumento es el dato que queremos asociar a dicha etiqueta. En este caso, a la primera etiqueta se le asocia el valor del límite recomendable de la aceleración (el límite por debajo del cual se considerado que se producirá mas ahorro de combustible), a la segunda etiqueta le asociamos el límite máximo permitido para la aceleración, por encima del cual el consumo es máximo.

Mediante el método ***obtenerVelocidadesUsuario*()** realizamos el mismo proceso que en el caso de la aceleración. Los datos incluidos en el objeto **Intent** referentes a los límites de velocidad se muestran en el **código fuente 29**:

```
intent.putExtra("VELOCIDAD_ECO", velocidadEcoString);
intent.putExtra("VELOCIDAD_MAX", velocidadConsMaxString);
```

Código Fuente 29. Introducción de datos de velocidad en el objeto Intent

Como en el caso de la aceleración, vamos a guardar los límites establecidos por el usuario en el objeto **Intent**. La primera etiqueta guarda el valor del límite recomendable para la velocidad, en la segunda etiqueta guardamos el límite máximo.

En el caso de que el usuario no haya elegido los valores para los límites descritos, utilizando para ello las barras de búsqueda de progreso, la aplicación asignará los límites establecidos por defecto y pedirá al usuario que confirme que desea emplear esos límites. Si el usuario seleccionó sus propios valores para los límites, el método

llamará a la actividad **AhorroCombustible** mediante la siguiente sentencia mostrada en **código fuente 30**:

```
startActivity(intent);
```

Código Fuente 30. Llamada al método startActivity()

Si el usuario desea emplear los límites por defecto deberá confirmar dicha decisión y entonces se llamará a esta misma sentencia para lanzar la siguiente actividad.

La actividad **AhorroCombustible** es lanzada a través del método onCreate(), este método actúa como el constructor de la clase, entre sus cometidos se encuentra asignar la vista a esta actividad y completar los atributos de la clase, en el **código fuente 31** podemos ver su pseudocódigo:

```
void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    reiniciarAtributosClase();

    Intent intent = getIntent();

    configurarAceleraciones(intent);
    configurarVelocidades(intent);

    configurarSonido();

    configurarObtenerCoordenadas();
}
```

Código Fuente 31. Método onCreate() de la actividad AhorroCombustible

Mediante el método ***getIntent*()** obtiene el objeto **Intent** que llamó a esta actividad. Los métodos ***configurarAceleraciones*()** y ***configurarVelocidades*()** extraen la información contenida en el **Intent** y completan los atributos de clase que guardan los límites para la conducción. A modo de ejemplo, en el **código fuente 32** podemos ver el pseudocódigo del método ***configurarAceleraciones*()**:

```
void configurarAceleraciones(Intent thisIntent) {

    String aceleracionEcoString =
thisIntent.getStringExtra("ACELERACION_ECO");

    aceleracionEco = Float.parseFloat(aceleracionEcoString);

    String aceleracionConsMaxString =
thisIntent.getStringExtra("ACELERACION_MAX");

    aceleracionConsMax =
Float.parseFloat(aceleracionConsMaxString);

    aceleracionIntermediaConsMax =
(aceleracionConsMax + aceleracionEco) / 2;
```

```
        aceleracionIntermediaEco = aceleracionEco / 2;
    }
```

Código Fuente 32. Método configurarAceleraciones()

Como se puede observar en el **código fuente 32** este método establece el valor de los atributos descritos en el apartado **3.2.4.7**, referente al llamado "**factorPendiente**". La forma de obtener los datos almacenados en el **Intent** es mediante el método ***Intent.getStringExtra*()**, incluyendo como argumento del método la etiqueta asociada al valor que queremos extraer.

El método ***configurarVelocidades*()** tiene como objetivo realizar las mismas operaciones que el método anterior, ocupándose en esta ocasión de los atributos referentes a los límites establecidos para la velocidad.

3.2.4.9 Intercambio de datos entre las actividades AhorroCombustible y AhorroResumen

Como se comentó en el apartado anterior, el segundo objeto **Intent** utilizado por la aplicación es lanzado desde la actividad **AhorroCombustible** para inicializar la actividad **AhorroResumen**.

La llamada a la actividad **AhorroResumen** es lanzada desde el método ***lanzarActividadResumen*()** de la clase **AhorroCombustible**. En el **código fuente 33** vemos su pseudocódigo:

```
private void lanzarActividadResumen() {

    float aceleracionMediaPositiva = 0;
    float aceleracionMediaNegativa = 0;
    float velocidadMedia = 0;
    float pendienteMedia = 0;

    if(numeroCoordenadasRecibidas != (float)0) {

        aceleracionMediaPositiva = aceleracionMediaPositivaSuma /
(numeroCoordenadasRecibidas - numeroCoordenadasRecibidasAceNegativas);

            aceleracionMediaNegativa =
 aceleracionMediaNegativaSuma /numeroCoordenadasRecibidasAceNegativas;

            velocidadMedia =
 velocidadMediaSuma / numeroCoordenadasRecibidas;

            pendienteMedia =
 pendienteMediaSuma / numeroCoordenadasRecibidas;
    }

     float conversorMsKmH = (float)3.6;

     int velocidadEcoKmH = (int)(velocidadEco * conversorMsKmH);

     int velocidadConsMaxKmH =
(int)(velocidadConsMax * conversorMsKmH);
```

```
    int velocidadMediaKmH = (int)(velocidadMedia * conversorMsKmH);
    int velocidadMaximaKmH =
(int)(velocidadMaxima * conversorMsKmH);

    Intent intent = new Intent(this, AhorroResumen.class);

    intent.putExtra("ACE_ECO_INTER", Float.toString(aceleracionEco));
    intent.putExtra("ACE_ECO_MAX",
Float.toString(aceleracionConsMax));

   //Reducimos a un decimal la aceleración positiva
   String aceleracionUnDecimal =
Float.toString(aceleracionMediaPositiva);
   aceleracionUnDecimal = aceleracionUnDecimal.substring(0,
aceleracionUnDecimal.indexOf(".") + 2);
   intent.putExtra("ACE_MEDIA_POS", aceleracionUnDecimal);

   //Reducimos a un decimal la aceleración negativa
   aceleracionUnDecimal = Float.toString(aceleracionMediaNegativa);
   aceleracionUnDecimal = aceleracionUnDecimal.substring(0,
aceleracionUnDecimal.indexOf(".") + 2);
  intent.putExtra("ACE_MEDIA_NEG", aceleracionUnDecimal);

  aceleracionUnDecimal = Float.toString(aceleracionMaximaPositiva);
   aceleracionUnDecimal = aceleracionUnDecimal.substring(0,
aceleracionUnDecimal.indexOf(".") + 2);
   intent.putExtra("ACE_MAXIMA_POS", aceleracionUnDecimal);

   aceleracionUnDecimal = Float.toString(aceleracionMaximaNegativa);
   aceleracionUnDecimal = aceleracionUnDecimal.substring(0,
aceleracionUnDecimal.indexOf(".") + 2);
   intent.putExtra("ACE_MAXIMA_NEG", aceleracionUnDecimal);

   intent.putExtra("VEL_ECO_INTER",
Integer.toString(velocidadEcoKmH));
   intent.putExtra("VEL_ECO_MAX",
Integer.toString(velocidadConsMaxKmH));
   intent.putExtra("VEL_MEDIA", Integer.toString(velocidadMediaKmH));
   intent.putExtra("VEL_MAXIMA",
Integer.toString(velocidadMaximaKmH));

   //Reducimos a un decimal la distancia total
   String distanciaUnDecimal = Float.toString(distanciaTotal);
   distanciaUnDecimal = distanciaUnDecimal.substring(0,
distanciaUnDecimal.indexOf(".") + 2);
   intent.putExtra("DISTANCIA", distanciaUnDecimal);

   //Reducimos a un decimal la pendiente media
   String pendienteUnDecimal = Float.toString(pendienteMedia);
   pendienteUnDecimal = pendienteUnDecimal.substring(0,
pendienteUnDecimal.indexOf(".") + 2);
   intent.putExtra("PENDIENTE", pendienteUnDecimal);

   startActivity(intent);
}
```

Código Fuente 33. Método lanzarActividadResumen() de la clase AhorroCombustible

Este método crea el objeto **Intent** que será enviado a la siguiente actividad e incluye en él los datos necesarios. En este caso, la actividad **AhorroResumen** sólo se encarga de mostrar los datos por lo que estos han de ser previamente procesados por la actividad actual.

Al principio del método calculamos los valores medios de velocidad, aceleración, deceleración (aceleración negativa) y también la pendiente media. Una vez obtenidos estos datos hay que transformar los datos de velocidad, que tienen como unidades m/s, a Km/h para que sean más fáciles de comprender por el usuario.

Con los datos necesarios ya calculados incluimos en el **Intent** los siguientes valores:

- **ACE_ECO_INTER**: El valor del límite recomendable sobre la aceleración establecido por el usuario.
- **ACE_ECO_MAX**: El valor del límite máximo permitido para la aceleración establecido por el usuario.
- **ACE_MEDIA_POS**: La aceleración media obtenida durante el viaje.
- **ACE_MEDIA_NEG**: La deceleración media obtenida durante el viaje.
- **ACE_MAXIMA_POS**: La aceleración máxima producida durante el viaje.
- **ACE_MAXIMA_NEG**: La deceleración máxima producida durante el viaje.

- **VEL_ECO_INTER**: El límite de velocidad recomendable fijado por el usuario.
- **VEL_ECO_MAX**: El límite de velocidad máximo fijado por el usuario.
- **VEL_MEDIA**: El valor de la velocidad media del vehículo durante el viaje.
- **VEL_MAXIMA**: El valor de la velocidad máxima alcanzada durante el viaje.

- **DISTANCIA**: La distancia total recorrida durante el viaje, en Km.

- **PENDIENTE**: La pendiente media de la vía por la que ha transcurrido el viaje.

El método termina con la llamada al método ***startActivity*()** que lanzará la actividad **AhorroResumen**.

Como en el caso del apartado anterior, para arrancar la siguiente actividad será llamado su método onCreate(), podemos ver dicho método en el **código fuente 34**:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.resumen_viaje);

    intent = getIntent();
    completarCampos();
 }
```

Código Fuente 34. Método onCreate() de la actividad AhorroResumen

El método ***completerCampos*()** es el encargado de extraer los datos del objeto **Intent** recibido, el **código fuente 35** muestra su pseudocódigo:

```
    private void completarCampos() {

        String aceleracionEcoInter =
intent.getStringExtra("ACE_ECO_INTER");
        String aceleracionEcoMax =
intent.getStringExtra("ACE_ECO_MAX");
        String aceleracionMediaPositiva =
intent.getStringExtra("ACE_MEDIA_POS");
        String aceleracionPos = "(" + aceleracionMediaPositiva + ", "
+ intent.getStringExtra("ACE_MAXIMA_POS") + ")";
        String aceleracionMediaNegativa =
intent.getStringExtra("ACE_MEDIA_NEG");
        String aceleracionNeg = "(" + aceleracionMediaNegativa + ", "
+ intent.getStringExtra("ACE_MAXIMA_NEG") + ")";

        String velocidadEcoInter =
intent.getStringExtra("VEL_ECO_INTER");
        String velocidadEcoMax = intent.getStringExtra("VEL_ECO_MAX");
        String velocidadMedia = intent.getStringExtra("VEL_MEDIA");
        String velocidadMaxima = intent.getStringExtra("VEL_MAXIMA");

        String distancia = intent.getStringExtra("DISTANCIA");
        String pendiente = intent.getStringExtra("PENDIENTE");

       (…)
}
```

Código Fuente 35. Método completarCampos() de la clase AhorroResumen

En esta ocasión, se ha suprimido la parte del código que asigna los valores extraídos del objeto **Intent** a cada una de las etiquetas de la vista correspondientes para cada valor.

Capítulo 4

Manuales

Durante este cuarto capítulo aprenderemos a instalar y a utilizar la aplicación desarrollada, para ello, se presentan dos manuales detallados. El manual de usuario mostrará cómo podemos utilizar todas las funcionalidades de nuestra aplicación, se describirán en detalle cada uno de los elementos que componen las distintas interfaces y veremos ejemplos de todas las pantallas. Por su parte, el manual de instalación nos enseñara a generar el paquete instalable de nuestra aplicación (paquete "**.apk**") y, posteriormente, nos indicará los pasos a seguir para instalar la plataforma en un dispositivo Android.

4.1 Manual de usuario

En este apartado vamos a aprender a utilizar la aplicación, este manual servirá para que cualquier persona que no sepa nada de la aplicación pueda utilizar completamente su funcionalidad. Para ello, describiremos las diferentes interfaces paso a paso, incluyendo imágenes reales de la aplicación a modo de ejemplo.

4.1.1 Pantalla de configuración

Vamos a comenzar nuestro manual de usuario por arrancar la aplicación, para ello, basta con pulsar sobre el icono de la misma (**figura 4.1**), debajo del cual también se indicará el nombre de la aplicación, en este caso **AhorroCombustible**.

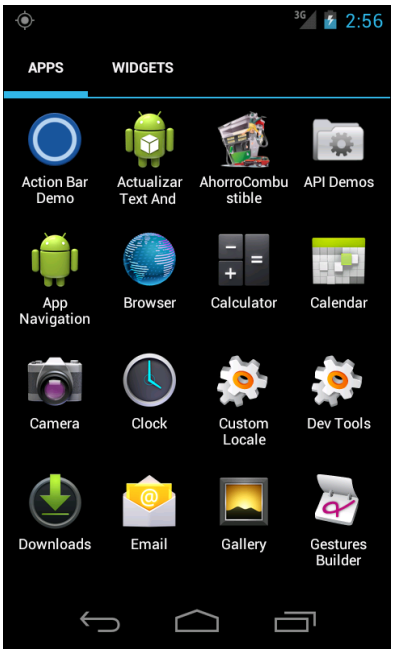


Figura 4.1. Icono de la aplicación

Una vez arrancada la aplicación se nos mostrará la pantalla de configuración como pantalla de inicio. En esta pantalla podremos establecer la configuración inicial sobre la que trabajará posteriormente nuestra plataforma. La aplicación nos va a permitir personalizar los límites que queremos respetar durante la conducción para conseguir realizar una conducción eficiente.

Se han establecido cuatro valores configurables en nuestra aplicación, dos para la aceleración y dos para la velocidad. Tanto para la velocidad como para la aceleración existe un límite recomendable y un límite máximo. El límite recomendable marcará la franja en la que consideramos que nuestro vehículo está haciendo un uso eficiente del combustible. La aplicación considerará que se está realizando una conducción eficiente cuando nos encontremos por debajo de este límite, no obstante, será tolerante cuando este se supere levemente, aunque indicará al usuario tal situación.

El límite máximo significa aquella frontera que no queremos cruzar bajo ningún concepto, ya que circular por encima de este límite supone un gasto de combustible intolerable. Cuando superemos este límite la aplicación lanzará avisos para que el usuario corrija esa situación. Los avisos son más contundentes en este caso pues pretenden conseguir que el usuario reaccione ante ellos para que cese la situación actual. Este tipo de avisos no sólo se emplearán después de superar este límite, se empezarán a lanzar cuando se supere el punto medio entre el límite recomendable y el límite máximo.

En el caso de la aceleración, los límites se emplean tanto para la aceleración como para la deceleración (frenadas, reducciones de velocidad). El valor para los límites elegido se empleará para las aceleraciones, el mismo valor con un signo negativo delante se empleará para las deceleraciones.

Se han establecido valores por defecto para los cuatro límites. En el caso de la velocidad, el valor escogido para el límite recomendable es de 60 Km/h, este valor implica que circulando en 5ª marcha estamos en la zona de menor consumo de nuestro vehículo, en el menor régimen de rpm posible. El valor del límite máximo se ha establecido en 100 Km/h, se ha elegido ese valor ya que es la franja de velocidad sobre la que el consumo se dispara.

Los límites por defecto para la aceleración se han fijado en 0.9 y 1.3 m/s^2, para la aceleración recomendable y la aceleración máxima respectivamente. Un turismo medio posee una aceleración máxima de 2.5 m/s^2, se ha convenido para el límite máximo la mitad de dicho valor. Para el límite recomendable se ha considerado un límite que no sea superado cuando se utilice el freno motor, implicando ese caso una desaceleración de -0.6 m/s^2 aproximadamente.

Gracias a la pantalla de configuración inicial podemos establecer los límites en función de nuestras necesidades, ya que, no es lo mismo conducir por una ciudad que hacer muchos kilómetros por una autovía, además, hay autovías que poseen límites de velocidad diferentes. Teniendo en cuenta lo anterior, una buena configuración de los

límites de velocidad consistiría en establecer como límite máximo el límite de velocidad de la calzada y como límite intermedio aquella velocidad con la que nos encontremos cómodos, tanto nosotros como el vehículo (porque este circule a un régimen de revoluciones moderado).

Para establecer los límites de aceleración tendremos en cuenta que cuanto más baja sea esta mayor será el ahorro de combustible, no obstante, también hay que tener en cuenta que a veces es necesario emplear una aceleración mayor para aumentar nuestra seguridad, este es el caso de las incorporaciones a vías de mayor velocidad.

La pantalla de configuración tendrá el aspecto mostrado en la **figura 4.2**:

Figura 4.2. Pantalla de configuración

En el límite superior de la pantalla mostrada en la **figura 4.2** podemos ver un texto explicativo referente a lo que espera la aplicación de nosotros. Observamos que la pantalla está dividida en dos secciones, una para la aceleración y otra para la velocidad, además, se incluye en la parte inferior de la pantalla el botón “Comenzar” cuya funcionalidad comentaremos un poco más adelante.

Antes de pulsar el botón “Comenzar” podemos realizar algunas modificaciones sobre los elementos de la interfaz gráfica. Encima de cada una de las barras de búsqueda de progreso existe una etiqueta que nos indica si se trata del límite máximo o del límite recomendado para la velocidad y la aceleración. Por defecto, la aplicación posee unos límites preestablecidos que ayudarán a los usuarios inexpertos a iniciarse en el tema de la conducción eficiente. En la misma etiqueta que indica la finalidad de la barra se indica el valor por defecto (“def=”) y las unidades de dicho valor, estas se encuentran descritas entre paréntesis.

Pulsando y arrastrando podemos modificar los valores de cada una de las barras de progreso. En el caso de la aceleración, el valor seleccionado aparecerá justo encima de la barra de progreso, junto al margen de la izquierda, este será de color amarillo para el límite recomendado y de color rojo para el límite máximo permitido. Los colores serán los mismos que se mostrarán en la siguiente pantalla de la aplicación cuando rebasemos dichos límites. A la misma altura que el valor, a la derecha de la pantalla, se muestra una etiqueta descriptiva del valor seleccionado, en ella se indica el tiempo que tardaría un vehículo en pasar de 0 a 100 Km/h si se le aplicara la aceleración seleccionada de manera constante. Esta segunda etiqueta también está pintada del color del valor del límite al que está asociada.

En el caso de la velocidad, los valores seleccionados en cada una de las barras se muestran encima de la propia barra, en esta ocasión, a la derecha de la interfaz. Al igual que ocurría con la aceleración, los valores para los límites de velocidad están pintados del mismo color con el que luego se mostrarán los avisos en la siguiente pantalla (amarillo para el límite recomendable y rojo para el límite máximo) siempre que rebasemos estos límites.

Dejando al margen las barras de búsqueda de progreso, esta pantalla contiene un menú que posee dos elementos seleccionables. Podremos acceder al menú pulsando el botón correspondiente de nuestro dispositivo, o bien, para versiones recientes de Android (posteriores al API 11), el menú aparecerá en la parte superior derecha de la pantalla (en forma de **ActionBar**).

Figura 4.3. Menú de la pantalla de configuración

En la **figura 4.3** vemos el menú de la pantalla en forma de **ActionBar**. Dentro del menú podemos seleccionar dos opciones, la opción “ayuda” y la opción “Salir”.

Si pulsamos la opción “ayuda” se mostrará una pantalla emergente que contiene información acerca de cómo utilizar esta primera pantalla (**figura 4.4**), además, también encontraremos una breve descripción de cada uno de los elementos relevantes de la misma. Podremos abandonar esta pantalla pulsando el botón “OK”.

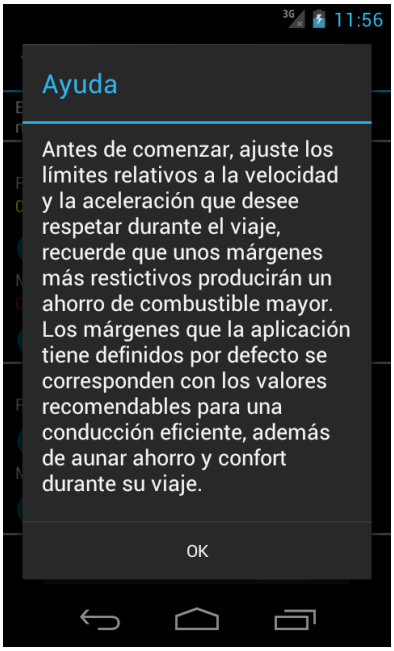


Figura 4.4. Pantalla de ayuda de configuración previa

Por el contrario, si en el menú de esta pantalla pulsamos la opción “Salir”, aparecerá un cuadro de confirmación que nos avisará de que estamos abandonando la aplicación (**figura 4.5**). Podemos elegir entre “Aceptar” y salir así de la aplicación, o bien, “Cancelar” en cuyo caso volveremos a la pantalla de configuración.

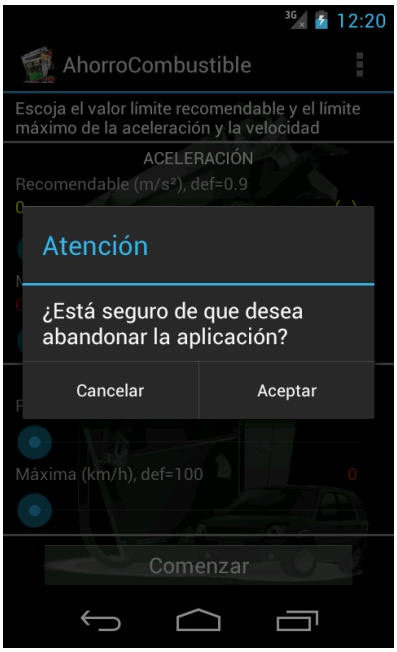


Figura 4.5. Pantalla confirmación abandonar aplicación

Volviendo al tema de la configuración de los límites, podemos elegir entre seleccionar nuestros propios límites para la aceleración y la velocidad, hacerlo sólo para la aceleración, sólo para la velocidad o para ninguno de ellos. Sobre aquellos límites que no modifiquemos se aplicarán los valores por defecto. Una vez hecho esto, podemos pulsar el botón “Comenzar”, de esta forma se iniciará la funcionalidad central de la aplicación. Cuando pulsemos este botón, si no hemos configurado los límites, se lanzará una pantalla de confirmación que nos indicará esta situación (**figura 4.6**), si pulsamos “Aceptar” en dicha pantalla pasaremos a la siguiente ventana con los valores por defecto establecidos para los límites, por el contrario, si pulsamos “Cancelar” volveremos a la pantalla de configuración y podremos modificar de nuevo los valores de cada límite.

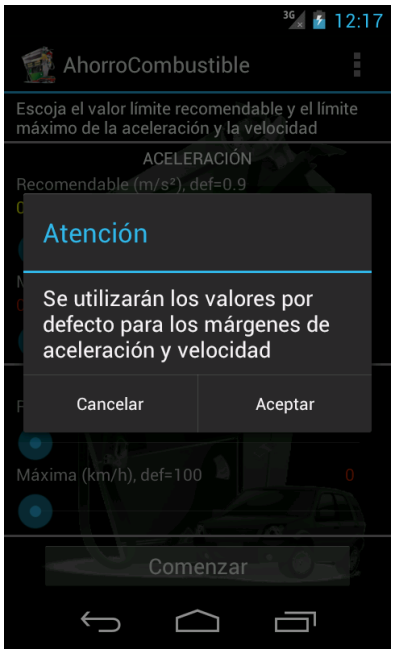


Figura 4.6. Pantalla de confirmación de los límites por defecto

Como se comentó anteriormente, la aplicación permite establecer nuestros límites para la velocidad y, sin en cambio, no modificar los límites de la aceleración y viceversa. La pantalla de confirmación anterior, en ese caso, indicará que se van a establecer los límites por defecto para los campos que no hayamos modificado. Por el contrario, si hemos seleccionado nuestros propios límites en las cuatro barras de progreso no se mostrará ninguna pantalla de configuración y pasaremos directamente a la siguiente ventana.

Hasta ahora, las imágenes de la aplicación mostradas hacen referencia a la posición vertical del dispositivo, si situamos el dispositivo en posición horizontal la pantalla de configuración tomará el siguiente aspecto (**figura 4.7**):

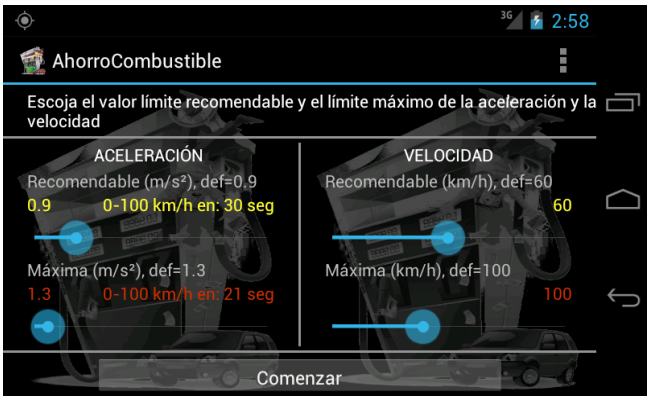


Figura 4.7. Pantalla de configuración en posición horizontal

A primera vista, podemos observar en la **figura 4.7** que la diferencia principal radica en la posición de los campos establecidos para la aceleración y la velocidad, en esta ocasión, los campos sobre aceleración y velocidad se encuentran uno al lado del otro, mientras que con el dispositivo situado en posición horizontal estos campos se muestran uno encima del otro.

4.1.2 Pantalla ahorro de combustible

Continuando con el manual de usuario de nuestra aplicación, vamos ahora a describir la considerada como pantalla principal de esta plataforma. Una vez establecida la configuración inicial, si el usuario pulsa el botón “Comenzar” se abrirá la siguiente ventana.

Antes de comenzar con la funcionalidad central del sistema, la plataforma hace una comprobación previa. La aplicación comprueba que el GPS del dispositivo este habilitado, en el caso de que el GPS esté activado la aplicación comenzará sin problemas, por el contrario, si la aplicación detecta que el GPS del dispositivo se encuentra deshabilitado mostrará una ventana emergente de aviso que le indicará al usuario tal situación (**figura 4.8**), entonces, la aplicación quedará a la espera hasta que el usuario habilite el GPS de su terminal. Esta es la ventana que se encontrará el usuario si su GPS está desactivado:

Figura 4.8. Pantalla de aviso de GPS desactivado

Resuelto el problema anterior, se mostrará ahora la pantalla principal de la plataforma, en la **figura 4.9** podemos ver su aspecto:

Figura 4.9. Pantalla principal de la aplicación

Esta pantalla (**figura 4.9**) se encuentra dividida en cuatro secciones. En la parte superior (en la vista vertical, a la izquierda en la vista horizontal) encontramos la imagen de un surtidor de gasolina antiguo, el color de fondo de esta imagen irá cambiando en función de la situación real del vehículo respecto a los límites establecidos. Se han establecido 5 colores que comprenden desde el verde hasta el rojo, pasando por el amarillo. Si rebasamos el límite máximo de aceleración o velocidad se

mostrará el color rojo como color de fondo. En el caso de rebasar el límite recomendable para la aceleración o la velocidad se mostrará el color amarillo. Por el contrario, si circulamos por debajo del límite recomendado, el color de fondo será el verde. Además, habrá un punto intermedio entre el color verde y el amarillo y otro punto intermedio entre el amarillo y el rojo.

Debajo de la imagen del surtidor (en el caso de la vista vertical, en la parte superior derecha en la vista horizontal) encontramos la sección encargada de mostrar al usuario la marcha de la caja de cambios en la que se recomienda circular, se recomendará una marcha u otra en función de la velocidad del vehículo. Además de mostrar en la pantalla dicha marcha recomendada se emitirán avisos sonoros indicando el cambio de marchas durante la conducción.

Debajo de la sección relativa a la marcha de la caja de cambios encontramos la sección referente a la situación actual del vehículo. En esta sección se muestra el valor de algunos datos relativos al movimiento del vehículo. Se han establecido cuatro etiquetas, cada una de ellas tiene asociado un valor que se encuentra a la derecha de esta. En este apartado, aparecen los valores actuales de aceleración, velocidad, pendiente y el porcentaje que varía el consumo del vehículo debido a la pendiente de la vía por la que circulamos.

Al igual que ocurre con el color de fondo de la imagen del surtidor, las etiquetas asociadas a la velocidad y a la aceleración también tienen asociado un color que depende de la situación actual frente a los límites establecidos en la pantalla anterior. Se trata de los mismos colores fijados para la imagen del surtidor. Cada etiqueta posee su propio color, por ejemplo, si la aceleración del vehículo se encuentra por debajo del límite recomendado esta se mostrará de color verde, en la misma situación, si la velocidad se muestra por encima del límite máximo se pintará su etiqueta de color rojo. Por tanto, cada etiqueta se irá mostrando del color adecuado en función de la situación actual, esto permite al usuario comprobar que límites ha superado con un sólo vistazo.

Por otro lado, el color de fondo de la imagen superior compartirá el color de la etiqueta que posea un color más grave (se considera más grave rebasar un límite máximo que un límite recomendado, y, a su vez, es más grave superar un límite recomendable que no superar ningún límite), por ejemplo, en el caso de antes, donde la aceleración estaba pintada de verde y la velocidad estaba pintada de rojo, el color de fondo de la imagen superior será el rojo.

En lo referente a la aceleración, los límites se aplican tanto sobre las aceleraciones como sobre las deceleraciones, por ejemplo, si el límite máximo para la aceleración se ha fijado en 1,3 m/s^2, suponiendo que realizáramos un frenazo brusco que provocara una aceleración inferior a -1,3 m/s^2 (una deceleración), se considerará rebasado dicho límite máximo.

Además de tener asociado un color, cada una de las situaciones posibles tiene asignado un sonido. Como ocurre con el color de fondo de la imagen, el sonido emitido

hará referencia a la situación más grave. A continuación, describiremos brevemente los sonidos emitidos en cada situación, con objeto de que puedan ser identificados durante la ejecución, asociaremos cada sonido con el color de fondo al que se refiere:

- Color **verde**: El sonido se compone de cuatro tintineos de una campanilla que ascienden en la escala tonal.
- Color **verde-amarillo**: En este caso, son tres tintineos que descienden en la escala tonal.
- Color **amarillo**: Tres sonidos cortos de un claxon.
- Color **amarillo-rojo** (debido a la **aceleración**): Un mensaje de voz con el aviso: "Modere la aceleración".
- Color **amarillo-rojo** (debido a una **deceleración**): Un mensaje de voz con el aviso: "Utilice el freno motor".
- Color **amarillo-rojo** (debido a la **velocidad**): Un mensaje de voz que dice: "Modere la velocidad".
- Color **rojo** (debido a una **aceleración**): Un mensaje de voz indicando: "Realice aceleraciones suaves".
- Color **rojo** (debido a una **deceleración**): Un mensaje de voz con el aviso: "Utilice desaceleraciones progresivas".
- Color **rojo** (debido a la **velocidad**): Un mensaje de voz que dice: "Reduzca la velocidad".

Debajo de los indicadores sobre la aceleración y la velocidad del vehículo, encontramos el indicador de la pendiente de la vía (en tanto por ciento) y el indicador de la variación en el consumo del vehículo debido a la pendiente y la velocidad (si la pendiente es cero la variación del consumo es nula, se considera el consumo normal del vehículo) a la que circulemos (indicado también en tanto por ciento).

Si el valor de la pendiente se muestra positivo indica una pendiente ascendente, se pintará además el valor de esta de color rojo, por el contrario, si el valor de la pendiente es negativo, esto significa que circulamos por una pendiente descendente y, en consecuencia, su valor estará pintado de verde.

La variación del consumo en función de la pendiente se calcula a partir de la propia pendiente y del valor de la velocidad actual, de esta forma se cuantifica, en porcentaje, la diferencia en el consumo de combustible producida entre movernos por una vía con una pendiente determinada respecto a circular por una vía con pendiente 0. Si esta etiqueta muestra un valor negativo quiere decir que el consumo se está reduciendo en ese porcentaje, en ese caso, el valor estará pintado de verde, si se muestra un valor positivo, este estará teñido de rojo, significa que el consumo se ha incrementado ese tanto por ciento.

En la parte inferior de la pantalla se muestran mensajes acerca de cómo actuar si hemos rebasado el límite máximo, tanto en el caso de la velocidad como en el caso de la aceleración, o bien, si hemos superado el límite intermedio situado entre el límite máximo y el límite recomendable, estos son los casos posibles:

- Color **amarillo-rojo** (debido a la **aceleración**): se mostrará el mensaje: "Modere La Aceleración".
- Color **amarillo-rojo** (debido a una **deceleración**): se mostrará el mensaje: "Utilice Freno Motor".
- Color **amarillo-rojo** (debido a la **velocidad**): se mostrará el mensaje: "Modere La Velocidad".
- Color **rojo** (debido a una **aceleración**): se mostrará el mensaje: "Reduzca Aceleración".
- Color **rojo** (debido a una **deceleración**): se mostrará el mensaje: "Use Frenadas Suaves".
- Color **rojo** (debido a la **velocidad**): se mostrará el mensaje: "Reduzca La Velocidad".

A parte de los mensajes mostrados hasta ahora, en el caso de que la aplicación detecte que el vehículo está detenido (velocidad = 0 Km/h) durante 30 segundos, mostrará al usuario indicaciones acerca de la recomendación de apagar el motor en paradas prolongadas (mayores de 60 segundos) para ahorrar combustible, los mensajes serán:

- Un mensaje tintado de azul en la parte inferior de la pantalla que muestra: "Stop >60s Motor Off".
- Un mensaje de voz que dicta: "En paradas largas apague el motor".

Podemos ver en la **figura 4.10** el aspecto que tendrá dicho mensaje.

Figura 4.10. Mensaje apagar motor en paradas prolongadas

A continuación, vamos a mostrar y describir algunos ejemplos de la pantalla principal durante la ejecución de la aplicación, hay que aclarar que para los ejemplos se

estableció la configuración "por defecto" en la pantalla de configuración, por tanto, los límites son:

- Aceleración recomendable = **0,9** m/s^2
- Aceleración máxima = **1,3** m/s^2
- Velocidad recomendable = **60** Km/h
- Velocidad máxima = **100** Km/h

En la **figura 4.11** encontramos el primer ejemplo:

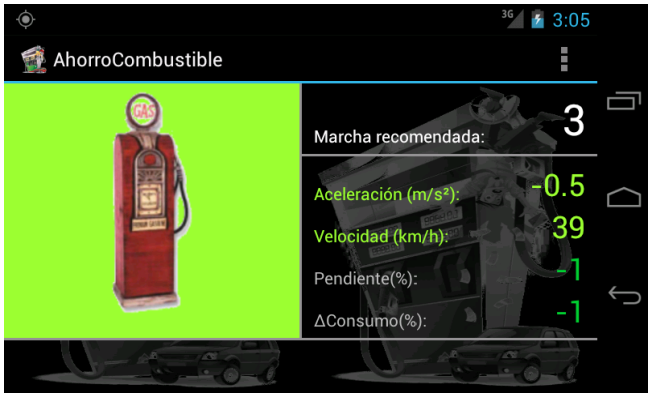


Figura 4.11. Ejemplo 1 pantalla principal

En este primer ejemplo, se está mostrando una captura de pantalla referente al caso de que el dispositivo se encuentre en posición horizontal, podemos empezar comprobando la marcha recomendada, observamos que la marcha recomendada en esta ocasión es la tercera marcha, esto es debido a que la velocidad actual es 39 Km/h. Los límites para las marchas se han establecido según las recomendaciones sobre conducción eficiente del IDAE:

- 1ª marcha: para reanudar la marcha hasta 3,5 Km/h
- 2ª marcha: desde 3,6 hasta 29 Km/h
- 3ª marcha: desde 30 hasta 39 Km/h
- 4ª marcha: desde 40 hasta 49 Km/h
- 5ª marcha: desde 50 Km/h

En este ejemplo, la aceleración es negativa (deceleración) lo que nos indica que se está reduciendo la velocidad del vehículo. Tanto la velocidad como la aceleración están pintadas de color verde-amarillo, esto significa que se encuentran entre el límite recomendable y la mitad de este. Para la aceleración, la mitad del límite recomendable sería 0,45 m/s^2, en este ejemplo, y para la velocidad este límite valdría 30 Km/h. Como la gravedad de ambos valores es la misma, la imagen muestra como color de fondo el color verde-amarillo.

También vemos en la **figura 4.11** que circulamos por una pendiente descendente (del -1 %), esto implica una reducción en el consumo del motor en un 1% respecto a circular a la misma velocidad por un llano.

Veamos ahora el siguiente ejemplo, contenido en la **figura 4.12**:

Figura 4.12. Ejemplo 2 pantalla principal

En este caso, el dispositivo se encuentra en posición vertical por lo que cambia ligeramente la distribución de la pantalla respecto al ejemplo 1. Al margen de eso, se observa que circulamos a una velocidad de 119 Km/h, en esta situación la aplicación nos recomienda circular en 5ª marcha. Además, esta velocidad rebasa el límite máximo permitido por lo que la etiqueta está pintada de rojo.

Por otro lado, la aceleración es negativa y tiene como valor -1,6 m/s^2, la aceleración, por tanto, también supera el límite máximo tolerable. En esta situación, la aplicación entiende que se ha actuado sobre el pedal de freno excesivamente y muestra en la parte inferior un mensaje indicando esta circunstancia al usuario, además, se reproducirá el mensaje de voz asociado a este caso. Ya que ambos límites máximos han sido superados el color de fondo de la imagen superior es el rojo.

Por último, podemos destacar los valores asociados a la pendiente. Esta vez, circulamos por una pendiente descendente más pronunciada que en el primer ejemplo, por el contrario, circulamos a mucha más velocidad que en el caso anterior así que la incidencia sobre el consumo total es baja, de esta forma, obtenemos el mismo valor referente a la variación del consumo debido a la pendiente.

A continuación, vamos a mostrar el último ejemplo de esta pantalla gracias a la **figura 4.13**:

Figura 4.13. Ejemplo 3 pantalla principal

En este último ejemplo, podemos observar en la **figura 4.13** que no hay aceleración ni pendiente alguna, es el caso de circular a una velocidad constante por una vía sin pendiente. En esta ocasión, el límite de velocidad máximo permitido está siendo superado, debido a esto, se muestran los colores y los mensajes que informan al usuario de esta situación (incluido el mensaje de voz que dicta: "Reduzca la velocidad").

Dejando atrás los ejemplos sobre la pantalla principal, podemos proseguir con la descripción de esta interfaz de usuario, vamos a centrarnos ahora en el menú de la misma. En la parte superior derecha encontramos tres cuadrados situados uno encima del otro, este es el indicador para desplegar el menú en forma de barra de acción, en APIs anteriores al 11 accederemos al menú pulsando el botón correspondiente. Este es el aspecto del menú una vez desplegado (**figura 4.14**):

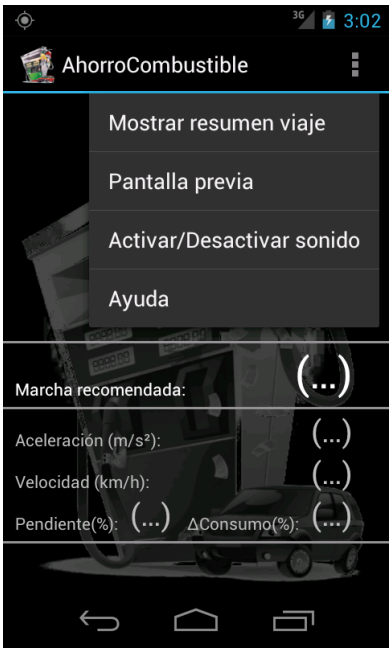


Figura 4.14. Menú de la pantalla principal

Como podemos observar en la **figura 4.14**, en esta ocasión, el menú se compone de cuatro elementos:

- "Mostrar resumen viaje": Nos muestra la pantalla que contiene el resumen del viaje.
- "Pantalla previa": Volvemos a la pantalla de configuración, perderemos los datos recogidos durante el viaje hasta ese instante.
- "Activar/Desactivar sonido": Se reproducirán los mensajes sonoros o no lo harán, por defecto el sonido se encuentra activado.
- "Ayuda": Se muestra una ventana emergente que incluye información relativa al interfaz actual.

La pantalla de ayuda se muestra en la **figura 4.15** y tiene el siguiente aspecto:

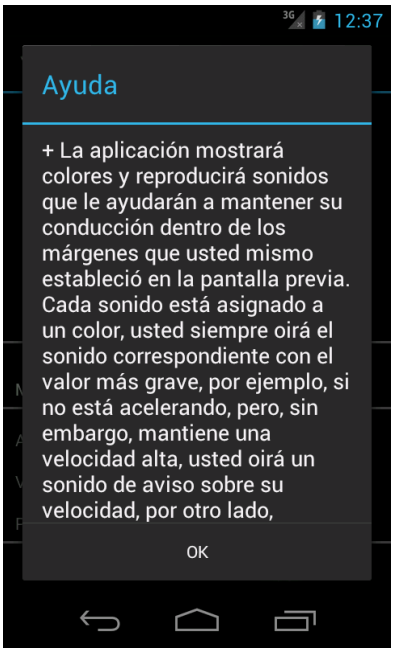


Figura 4.15. Ventana de ayuda de la pantalla principal

4.1.3 Pantalla resumen del viaje

Cuando el usuario pulse la opción “Mostrar resumen viaje” del menú asociado a la pantalla principal de la aplicación, accederemos a la pantalla que contiene el resumen del viaje, esto comprende todos los datos obtenidos desde que el usuario arranca la pantalla principal (con el GPS activado) hasta el momento en que es lanzada esta pantalla. Cada vez que el usuario vuelva a la pantalla de configuración se borrarán todos los datos anteriores.

Este es el aspecto de la pantalla de resumen (**figura 4.16**), con el dispositivo situado en posición vertical:

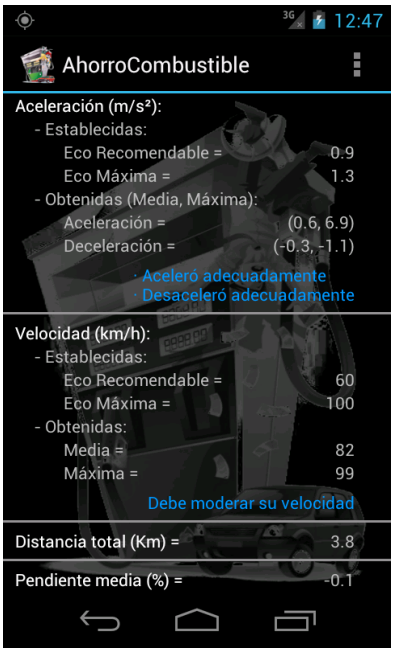


Figura 4.16. Pantalla resumen del viaje en posición vertical

Este es el aspecto de la pantalla resumen cuando el dispositivo se encuentra en posición horizontal (**figura 4.17**):

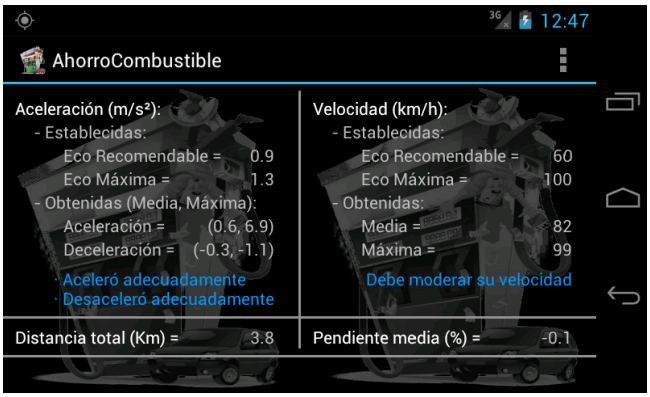


Figura 4.17. Pantalla resumen del viaje en posición horizontal

En las pantallas mostradas (**figuras 4.16 y 4.17**) se observa que ambas contienen los mismos datos, esto es debido a que se imprimieron en el mismo momento con la única diferencia de la posición del terminal.

Como se muestra en las imágenes, ambas pantallas están divididas en cuatro campos: aceleración, velocidad, distancia total y pendiente media.

En el campo referente a la aceleración nos encontramos varios datos, todos ellos poseen como unidades m/s^2. En la primera parte observamos los límites establecidos por el usuario en la pantalla de configuración. Debajo de los límites establecidos al inicio, encontramos los resultados obtenidos durante el viaje. Los datos se muestran entre paréntesis, en la parte izquierda del paréntesis se muestra el valor medio de ese dato obtenido durante el viaje, en la parte derecha encontramos el valor máximo ocurrido.

En la parte inferior del campo reservado para la aceleración encontramos dos mensajes que sirven al usuario como recomendaciones para mejorar su conducción. En este caso encontramos un mensaje para la aceleración y otro para la deceleración. Los mensajes se establecen en función de los resultados medios obtenidos, como en este caso los valores medios se encuentran por debajo del límite recomendable para la aceleración, la aplicación nos indica que nuestra conducción se llevo a cabo correctamente.

En el campo referente a la velocidad, encontramos también en la parte superior los valores establecidos para los límites en la pantalla de configuración. Debajo de estos límites encontramos la velocidad media obtenida durante el viaje y la velocidad máxima. Como ocurre con la aceleración, en la parte inferior se muestra un mensaje que hace alguna indicación al usuario referente a su conducción. En este caso la velocidad media calculada se encuentra por encima del límite recomendable y por debajo del límite máximo, la aplicación indica al usuario que debe moderar la velocidad en futuras ocasiones.

A parte de la aceleración y la velocidad, también se muestra en esta pantalla la distancia total recorrida durante el viaje, en Kilómetros.

Por último, la pantalla resumen nos muestra el valor medio de la pendiente que hemos encontrado durante la ejecución, el valor de la pendiente se muestra en tanto por ciento.

Por otro lado, esta pantalla contiene un menú con dos elementos, el elemento “Información” y el elemento “Volver”, podemos observar el aspecto de dicho menú gracias a la **figura 4.18**:

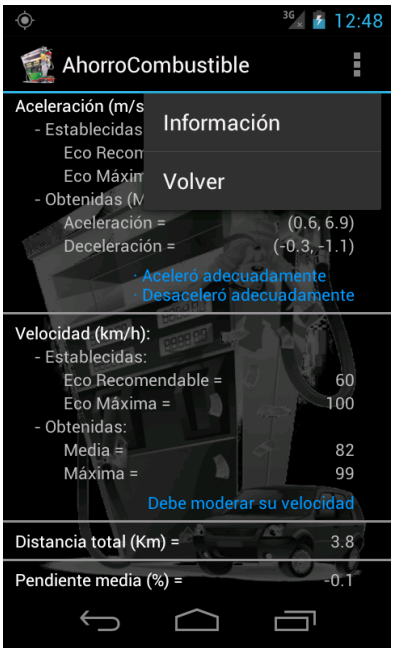


Figura 4.18. Menú de la pantalla resumen

El elemento “Volver” nos conduce a la pantalla principal de la aplicación, donde seguirá corriendo la aplicación normalmente, como lo hacía antes de entrar en la pantalla de resumen.

El elemento “Información” muestra una ventana emergente con información relativa a esta pantalla y con alguna de las recomendaciones más importantes del IDAE sobre conducción eficiente (**figura 4.19**), esta es una parte de su contenido:

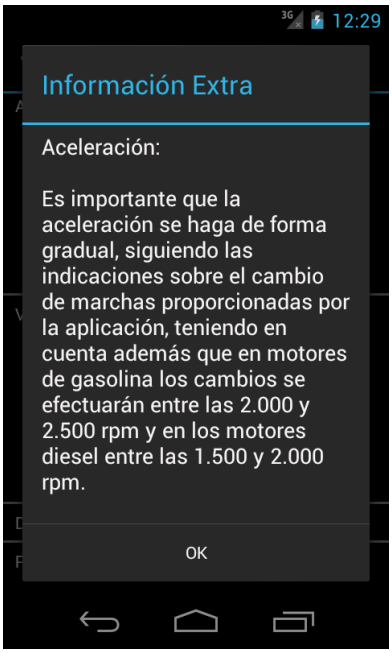


Figura 4.19. Información extra de la pantalla resumen

4.2 Manual de instalación

En esta sección aprenderemos todo lo necesario para instalar nuestra aplicación en cualquier dispositivo compatible con la misma. Empezaremos por mostrar cómo podemos generar el paquete de instalación (.apk) de la aplicación desde Eclipse. Una vez generado, este paquete podrá ser publicado en Google Play para su difusión entre la comunidad Android.

4.2.1 Generar el paquete .apk de la aplicación apto para Google Play

Durante el desarrollo de la aplicación con Eclipse, cuando compilamos nuestra aplicación Android se genera el paquete **.apk** de la aplicación en “modo depuración” (del inglés “debug mode”) y se le añade automáticamente una “llave” (del inglés “key”) de depuración. El paquete generado de esta manera es completamente instalable y

ejecutable, sin embargo, este paquete no es apto para su distribución en Google Play. **[18]**

Para generar con Eclipse un paquete compatible con "Google Play" es necesario que este se encuentre firmado correctamente, para ello, seguiremos los siguientes pasos:

1. Pulsamos con el botón derecho del ratón sobre el proyecto.
2. Seleccionamos la opción "Export…".
3. En la siguiente pantalla (**figura 4.20**), marcamos "Export Android Application".

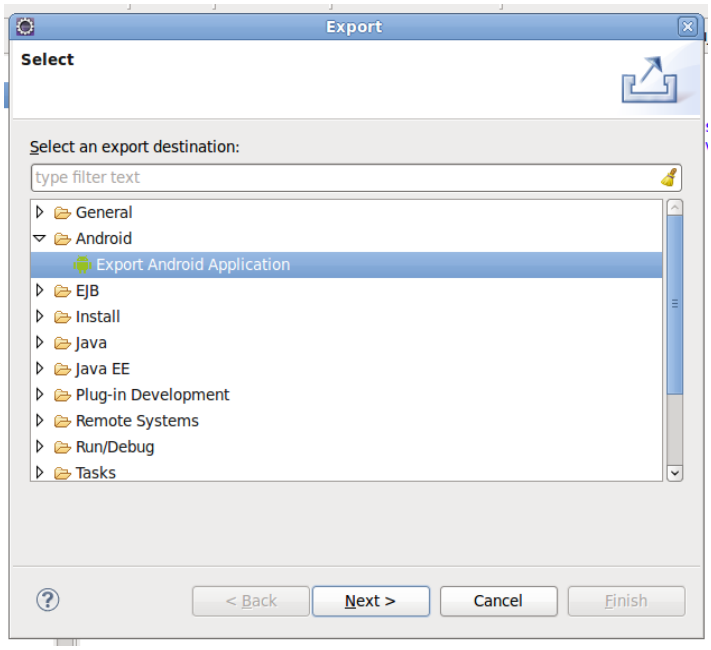


Figura 4.20. Generar APK con Eclipse, paso 1

4. En la pantalla emergente (**figura 4.21**) podemos seleccionar el proyecto deseado, si no es el que aparece seleccionado directamente.
5. Pulsamos "Next >"

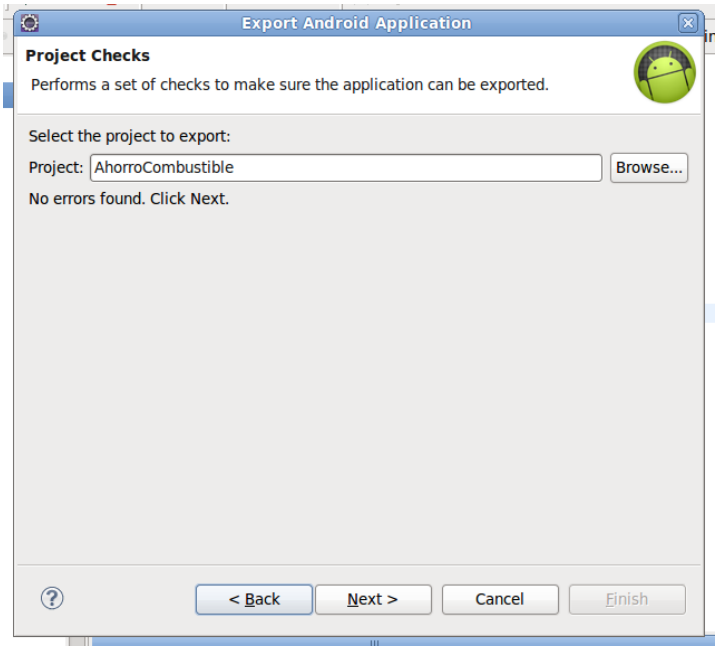


Figura 4.21. Generar APK con Eclipse, paso 2

6. Ahora debemos firmar la aplicación (**figura 4.22**). Si es la primera vez que exportamos una aplicación, pulsaremos en "Create new keystore", seleccionamos un directorio para almacenar las claves y establecemos una contraseña.

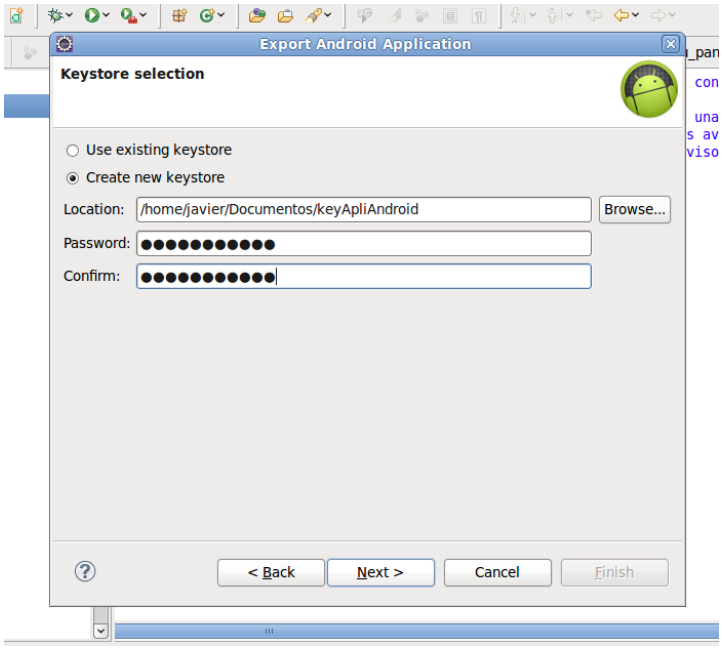


Figura 4.22. Generar APK con Eclipse, paso 3

7. En la siguiente pantalla (**figura 4.23**), terminaremos de completar los campos con nuestra información.

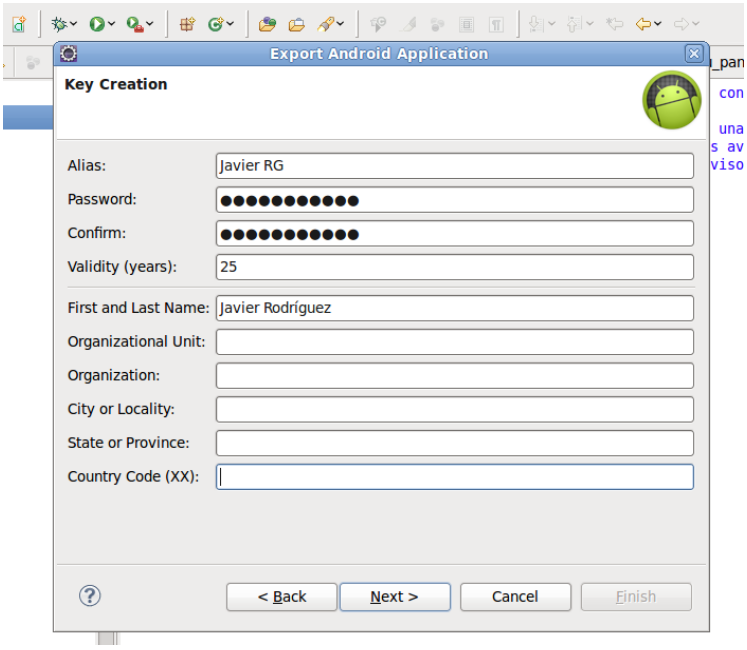


Figura 4.23. Generar APK con Eclipse, paso 4

8. En la última pantalla (**figura 4.24**), podemos seleccionar el directorio destino donde se almacenará el archivo .apk.

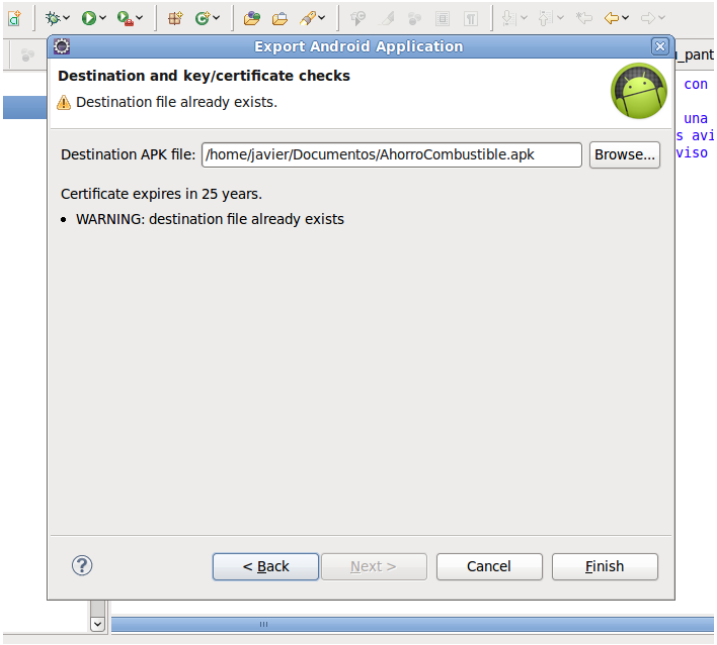


Figura 4.24. Generar APK con Eclipse, paso 5

4.2.2 Manual de instalación

El primer paso para instalar nuestra aplicación en un dispositivo Android es copiar el archivo .apk a dicho terminal.

En el caso de que la aplicación se encuentre publicada en “Google Play”, podremos descargar dicho archivo desde allí y automáticamente se instalará la aplicación en el terminal.

Por otro lado, si poseemos previamente el archivo .apk, bastará con copiar este archivo al Smartphone. Podemos realizar dicha operación de muchas maneras, puede ser a través del puerto USB del dispositivo, a través de una tarjeta de memoria (una SD-card), Bluetooth o utilizando algún programa que establezca una conexión con el dispositivo mediante la red de datos (como puede ser AirDroid, este se encuentra disponible de manera gratuita en “Google Play”).

Una vez copiado el archivo .apk en el dispositivo, mediante un explorador de archivos, buscaremos la ubicación donde hemos guardado el archivo. Cuando lo hallemos, pulsaremos sobre este y posteriormente pulsaremos “Instalar”, como se muestra en la **figura 4.25**.

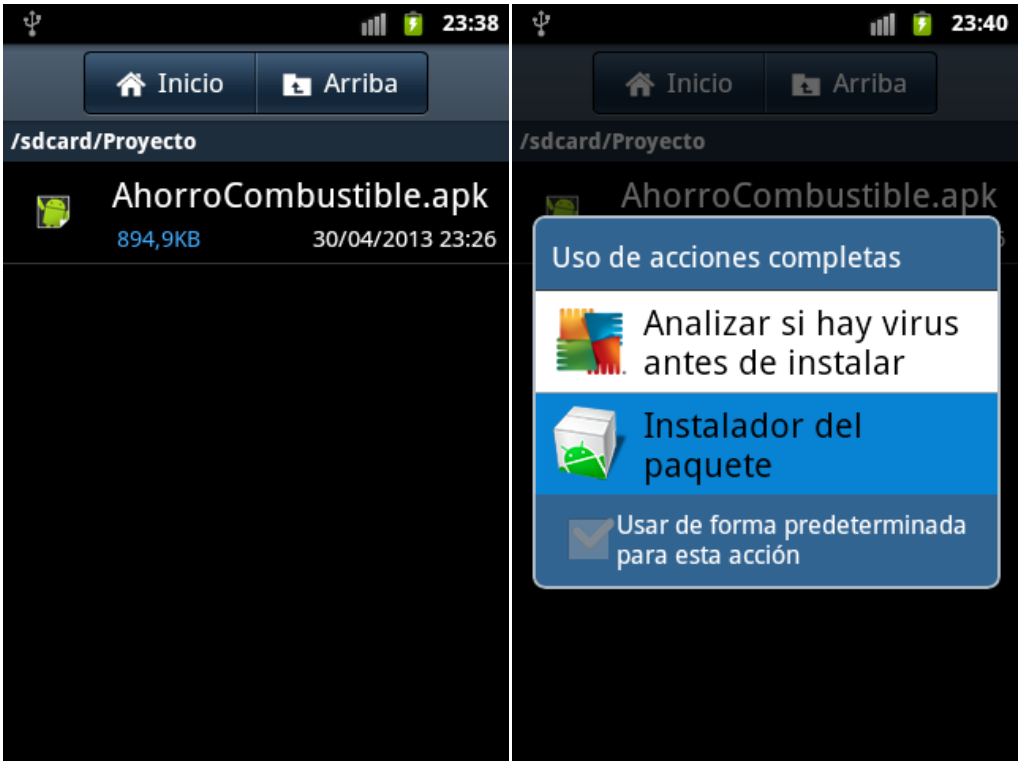


Figura 4.25. Instalación de la aplicación en el dispositivo, paso 1

Comenzará entonces la instalación de la aplicación, acto seguido, aparecerá una pantalla en la que se nos solicitarán los permisos necesarios para ejecutar la aplicación, en nuestro caso, se solicitarán permisos sobre la localización del dispositivo (**figura 4.26**).

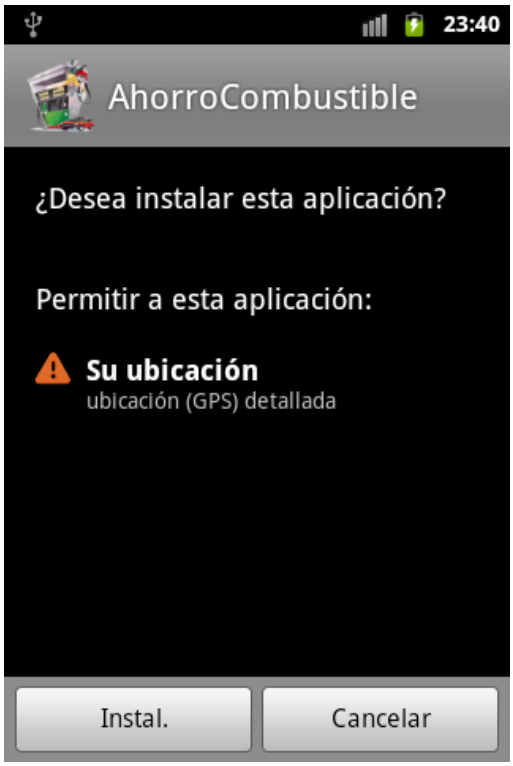


Figura 4.26. Instalación de la aplicación en el dispositivo, paso 2

Para poder completar la instalación es necesario confirmar estos permisos solicitados, en caso contrario, la instalación de la aplicación terminará y no podremos ejecutar posteriormente dicha aplicación.

Una vez terminado el proceso de instalación, podremos lanzar la aplicación siempre que queramos, recordando que el dispositivo GPS del terminal ha de estar activado para el correcto funcionamiento de esta.

Capítulo 5

Resultados

En este capítulo someteremos a nuestra aplicación a diversas pruebas controladas cuyos resultados demostrarán el buen funcionamiento y la efectividad del sistema desarrollado en el presente proyecto, además de comprobar que se cumplen los requisitos establecidos en el tema 3.

Para ello, realizaremos tres fases de pruebas distintas. La primera de las fases se llevará a cabo mediante el emulador proporcionado por el kit de desarrollo de Android. La segunda de ellas consistirá en evaluar la precisión de los cálculos en una situación real, comparando los datos obtenidos mediante el GPS con los datos ofrecidos en los indicadores del vehículo. La última fase consistirá en comparar el consumo obtenido realizando dos veces el mismo recorrido, una de las veces siguiendo las indicaciones de la aplicación y la otra conduciendo normalmente, siguiendo la tendencia del resto del tráfico.

5.1 Fase de pruebas con el emulador

El kit de desarrollo de Android (Android SDK) incluye un emulador de un dispositivo Android que nos permite probar nuestra aplicación, aunque este emulador no es un dispositivo completo (posee algunas carencias) podemos suplir algunas de sus carencias mediante nuestro propio código java y así probar todas las funcionalidades de nuestra aplicación.

Para probar como gestiona los datos de localización nuestra aplicación podemos hacer uso del siguiente comando en el emulador:

geo fix <Longitud> <Latitud>

Este comando simula la llegada de un objeto **Location** proveniente del GPS. Para utilizar este comando deberemos estar conectados, mediante el protocolo **telnet** al puerto donde hemos configurado que escuche nuestro emulador (en nuestro caso es el puerto 5554). Para conectarnos a dicho puerto, mediante la consola del sistema operativo del computador, primero deberemos situarnos en la dirección donde tengamos guardado el directorio ***/android-sdk-linux/tools***, una vez situados en dicho directorio escribiremos (el emulador de Android deberá estar corriendo):

telnet localhost 5554

Desde ese instante estaremos conectados al emulador y podremos introducir comandos para provocar diferentes situaciones en dicho emulador, como puede ser introducir una coordenada, simular una llamada telefónica u otros eventos.

Para nosotros es importante comprobar que la aplicación realiza correctamente los cálculos sobre velocidad, aceleración, distancia, etc. Además de comprobar que se

realizan correctamente los cálculos también comprobaremos que el sistema aplica correctamente los límites establecidos por el usuario para la velocidad y la aceleración.

Estableceremos varios casos diferentes donde una serie de coordenadas genere diversas situaciones controladas y así comprobaremos si los resultados obtenidos de la aplicación se corresponden con los datos introducidos.

Debido a que queremos conocer a priori la situación para poder valorar a posteriori dicha situación mostrada por la aplicación hemos de aprender a manejar correctamente el uso de coordenadas. Sabemos que las coordenadas están compuestas por latitud y longitud, normalmente ambos valores se expresan en grados y sabemos que tienen el origen de coordenadas en el punto donde se cruzan el Ecuador y el meridiano de Greenwich.

Lo primero que nos interesa es saber la distancia, en línea recta, entre dos coordenadas situadas en la superficie de la Tierra, para calcular la distancia entre la coordenada A y la coordenada B es conocida la siguiente fórmula:

Distancia (A, B) = Radio Tierra + acos [cos(LatA) · cos(LatB) · cos(LonB -LonA) + sen(LatA) · sen(LatB)]

En la fórmula anterior aparece el radio de la Tierra, emplearemos para el cálculo de la distancia el radio medio de la Tierra, que es 6.371 Km. Además, cabe destacar que los valores se han de utilizar en radianes, no en grados, para pasar de grados a radianes la relación es la siguiente:

radianes = grados · π / 180

Con el objetivo de simplificar los cálculos sobre la distancia entre coordenadas fijaremos el valor de la Longitud en 0º y será el valor de la Latitud el que iremos variando para generar coordenadas separadas entre sí la distancia que deseemos.

Además, ya que es muy pesado calcular para cada coordenada la fórmula anterior, estableceremos una distancia de referencia de 1º en la Latitud y sobre esta distancia, aplicando proporciones, calcularemos las distintas coordenadas. El cálculo de esta distancia de referencia se muestra a continuación:

- *Coordenada A = (0.0, 0.0)*
- *Coordenada B = (0.0, 1.0)*

Descomponiendo la fórmula en factores simples:

- *cos (LatA) = cos (0 · π / 180) = 1*
- *cos (LatB) = cos (π / 180) = 0.9998*
- *cos (LonB – LonA) = 1*
- *sen (LatA) = 0*
- *sen (LatB) = sen (π / 180) = 0.01745*

Entonces:

Distancia (A, B) = 6371 · acos [1 · cos (π / 180) · 1 + 0] = 111,2 Km = 111.200 m

La distancia obtenida es muy grande para el uso que vamos a hacer nosotros de las coordenadas, por ello, vamos a calcular a partir del resultado anterior, mediante una relación sencilla, la distancia entre dos coordenadas separadas 0,001º de latitud, será entonces dicha distancia:

Distancia entre (0.0, 0.0) y (0.0, 0.001) = 0,001 · 111.200 / 1 = 111,2 m

5.1.1 Primer caso de pruebas en el emulador

Las condiciones establecidas para este primer caso de pruebas son:

- Velocidad constante de 40 Km/h.
- Límites por defecto para la aceleración y la velocidad.
- Sin pendientes.

Para poder llevar a cabo las pruebas con el emulador se ha generado un **Script** en cada caso que introduce automáticamente las coordenadas en el emulador, en este primer caso el **Script** empleado se muestra en la **figura 5.1**:

```
#!/bin/bash
#Script lanza coordenadas gps
cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0005"
 sleep 5
 echo "geo fix 0.0 0.0010"
 sleep 5
 echo "geo fix 0.0 0.0015"
 sleep 5
 echo "geo fix 0.0 0.0020"

(…)

echo "geo fix 0.0 0.0165"
sleep 5
 echo "geo fix 0.0 0.0170"
sleep 5
echo "geo fix 0.0 0.0175") | telnet localhost 5554
```

Figura 5.1. Script primer caso de pruebas en el emulador

Este Script se conecta mediante telnet al emulador de Android e introduce las coordenadas deseadas. Entre cada una de las coordenadas espera un tiempo concreto (5 segundos) que permite simular velocidad y aceleración, a través de las coordenadas

introducimos el espacio recorrido y con el tiempo entre ellas podemos calcular la velocidad del vehículo.

Podemos observar en el código del Script que al principio hay dos coordenadas iguales, esto nos va a servir para inicializar a 0 la velocidad y la aceleración de la aplicación. Las coordenadas están separadas entre sí 0.0005º, esto supone una distancia entre ellas de:

Distancia entre coordenadas = 0.0005 · 111,2 / 0.001 = ***55,6 m***

Como el tiempo entre ellas es 5 segundos:

Velocidad *= distancia / tiempo = 55,6 / 5 = 11,12 m/s =* ***40*** *Km/h*

Con los datos de entrada preparados, ya podemos lanzar la aplicación. En la **figura 5.2** se muestra la pantalla de configuración para el presente caso de prueba:

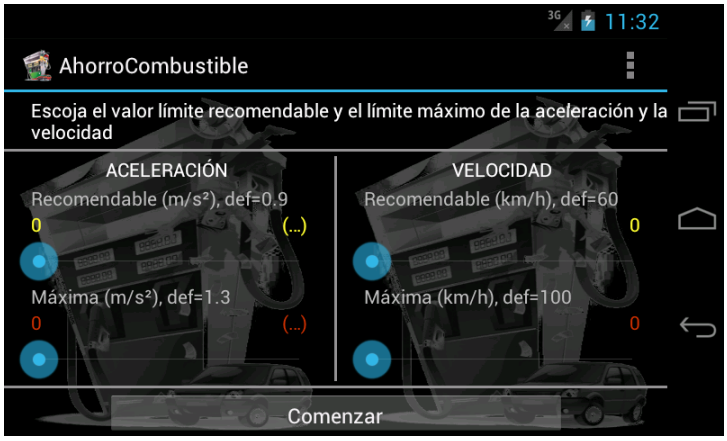


Figura 5.2. Caso de pruebas 1, pantalla de configuración

Como se puede observar en la **figura 5.2**, se ha establecido la configuración inicial con los valores que tiene asignados la aplicación por defecto para todos los límites.

Durante la ejecución de la plataforma y con el Script corriendo se observa la siguiente pantalla en el emulador (**figura 5.3**):

Figura 5.3. Caso de pruebas 1, pantalla principal

En la **figura 5.3** encontramos que no existe aceleración alguna, no hay pendiente y la velocidad permanece constante en 39 Km/h. Circulando a esta velocidad la aplicación nos recomienda engranar la 3ª marcha, recordemos que el límite inicial para la 4ª marcha empieza en 40 Km/h.

Respecto a los límites establecidos, podemos observar que la aceleración se ha pintado de verde, debido a que esta se encuentra por debajo del límite recomendable. Por otro lado, la velocidad se ha teñido de verde-amarillo, ya que el valor de esta se encuentra por encima de la mitad del límite recomendable. Como el caso de la velocidad es el más grave, el color de fondo de la aplicación se pinta del color de la velocidad.

En la **figura 5.3** observamos que el valor de la velocidad es 39 Km/h frente a los 40 Km/h previstos en el Script, esta pequeña diferencia se debe, en parte, a la aproximación que hace el programa de los números decimales. Recordemos que el programa trabaja en m/s con decimales pero a la hora de mostrar el dato al usuario utiliza un número entero con las unidades Km/h, por eso se pierde precisión en este dato.

Por otro lado, la pequeña diferencia encontrada frente al valor esperado también se puede deber a la diferencia entre la forma de calcular la distancia que usamos nosotros y la forma concreta en la que lo hace el método ***Location.distanceTo(Location)*** del API de Android, encargado de calcular las distancias entre coordenadas en la aplicación, además, ya se comentó en un apartado anterior que este método es un poco impreciso.

Una vez ha terminado el Script podemos comprobar los datos ofrecidos en la pantalla resumen de la plataforma, podemos observar dicha pantalla en la **figura 5.4**:

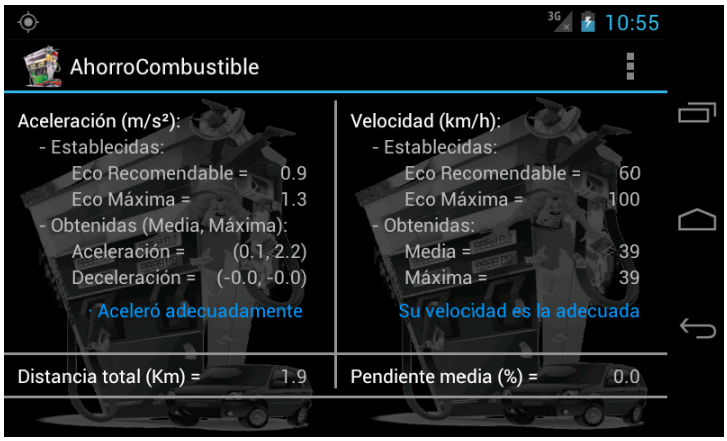


Figura 5.4. Caso de pruebas 1, pantalla resumen

En la pantalla resumen contenida en la **figura 5.4** podemos comprobar varias cosas. Por un lado, observamos que la velocidad media y la máxima se corresponden con lo esperado.

Por otra parte, se ha registrado una aceleración durante la llegada de las coordenadas. Esto se debe a que las coordenadas empiezan con velocidad de 0 Km/h y

después de algunas coordenadas cambian a la velocidad final, esa aceleración es la que se muestra en esta pantalla resumen.

Por último, también podemos observar el dato de la distancia total, en este caso 1.900 m. Dividiendo este dato por el número de coordenadas, en total 34, el resultado es la distancia entre coordenadas, aproximadamente, recordemos que hay pérdida de precisión al convertir valores entre diferentes unidades:

Distancia entre coordenadas = 1.900 / 34 ≈ ***56*** *m*

5.1.2 Segundo caso de pruebas en el emulador

En este segundo caso cambiaremos un poco las condiciones para observar el comportamiento de nuestra aplicación, estas condiciones son:

- Velocidad constante de 120 Km/h.
- Límites establecidos en los valores por defecto.
- Sin pendiente.

La **figura 5.5** contiene el **Script** empleado para el segundo caso de pruebas:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0015"
 sleep 5
 echo "geo fix 0.0 0.0030"
 sleep 5
 echo "geo fix 0.0 0.0045"
 sleep 5

(…)

 echo "geo fix 0.0 0.0495"
sleep 5
echo "geo fix 0.0 0.0510"
sleep 5
echo "geo fix 0.0 0.0525") | telnet localhost 5554
```

Figura 5.5. Script Segundo caso de pruebas en el emulador

Como en el caso anterior, el tiempo entre coordenadas se mantiene, lo que cambia es la distancia entre coordenadas (0.0015º), de esta forma variamos la velocidad. La distancia entre coordenadas en esta ocasión es:

Distancia entre coordenadas = 0.0015 · 111,2 / 0.001 = ***166,8 m***

La velocidad obtenida mediante estas coordenadas debe ser:

***Velocidad** = 166,8 / 5 = 33,36 m/s = **120,1** Km/h*

La pantalla de configuración, en este caso, es la misma que en el caso anterior, de modo que no la vamos a mostrar, empezaremos mostrando directamente la pantalla resultante de la aplicación cuando ejecutamos el Script anterior (**figura 5.6**):

Figura 5.6. Caso de pruebas 2, pantalla principal

En esta ocasión, observamos que los datos ofrecidos por la aplicación se corresponden con los esperados. No hay aceleración ni pendiente, la velocidad es aproximadamente 120 Km/h y las señales mostradas por la aplicación se corresponden a dicha circunstancia.

La pantalla resumen resultante al ejecutar este caso de pruebas la podemos ver en la **figura 5.7**:

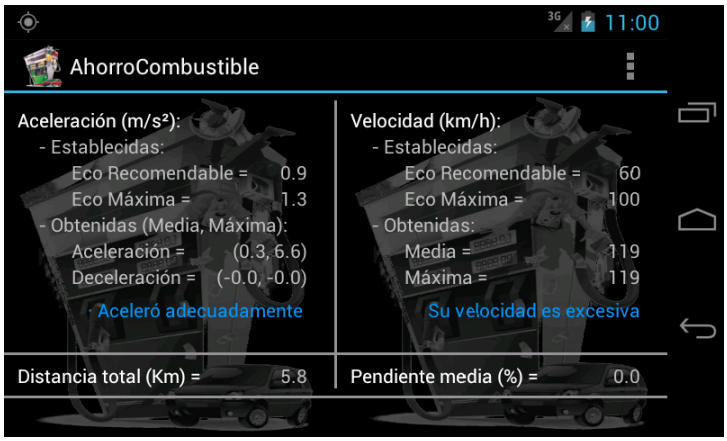


Figura 5.7. Caso de pruebas 2, pantalla resumen

Se puede observar en la **figura 5.7** que los datos mostrados son consistentes con la situación generada para la prueba de la aplicación.

5.1.3 Tercer caso de pruebas en el emulador

En este tercer caso vamos a probar a modificar los límites establecidos para la velocidad pero no vamos a modificar los de la aceleración. Las condiciones iniciales para esta prueba serán entonces:

- Velocidad constante de 80 Km/h.
- Límites sobre aceleración en sus valores por defecto.
- Límite recomendable de velocidad = 70 Km/h, límite máximo permitido para la velocidad = 120 Km/h.
- Sin pendiente.

La pantalla de configuración tiene el siguiente aspecto (**figura 5.8**) cuando establecemos los límites mencionados en las condiciones iniciales:

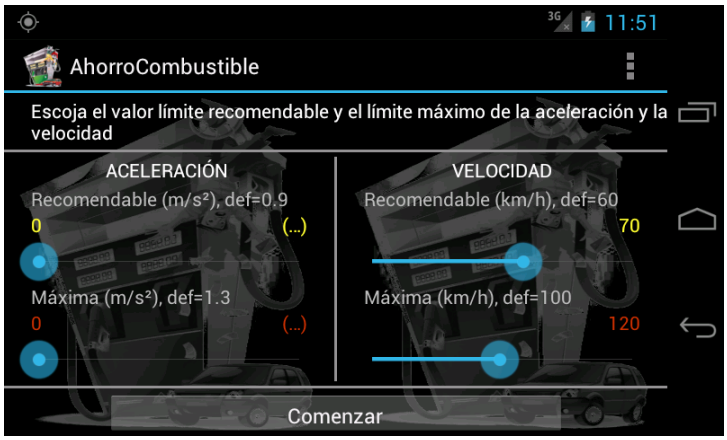


Figura 5.8. Caso de pruebas 3, pantalla de configuración

En la **figura 5.9** se detalla el Script generado para este tercer caso de pruebas, tendrá el siguiente aspecto:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.001"
 sleep 5
 echo "geo fix 0.0 0.002"
 sleep 5
 echo "geo fix 0.0 0.003"
 sleep 5
 echo "geo fix 0.0 0.004"
 sleep 5

(…)

 echo "geo fix 0.0 0.032"
```

```
 sleep 5
 echo "geo fix 0.0 0.033"
sleep 5
 echo "geo fix 0.0 0.034"
sleep 5
 echo "geo fix 0.0 0.035") | telnet localhost 5554
```

Figura 5.9. Script tercer caso de pruebas en el emulador

En esta ocasión, las coordenadas mostradas en la **figura 5.9** están separadas entre sí 0.001º de Latitud. Siguiendo el mismo procedimiento que en los casos anteriores, la distancia en metros entre las coordenadas es de **111,2** metros. Por lo tanto, la velocidad calculada a partir de estas coordenadas es de **80,8** Km/h. Observemos ahora la pantalla principal de nuestra aplicación (**figura 5.10**) cuando aplicamos el Script anterior al emulador:

Figura 5.10. Caso de pruebas 3, pantalla principal

Debido a que hemos modificado los límites, ahora la aplicación se comporta de manera más permisiva con la velocidad que en los dos primeros casos. La velocidad y el fondo de la imagen del surtidor se pintan de color amarillo ya que la velocidad apenas rebasa el límite recomendable.

Del mismo modo, en la parte inferior no aparecen mensajes referentes a una reducción de la velocidad, además, en este caso, el sonido que reproduce la aplicación a modo de aviso es el de tres golpes de claxon.

Veamos ahora la pantalla de resumen (**figura 5.11**), esta indicará los nuevos límites establecidos para la velocidad, además de indicar el resto de información que ya conocemos.

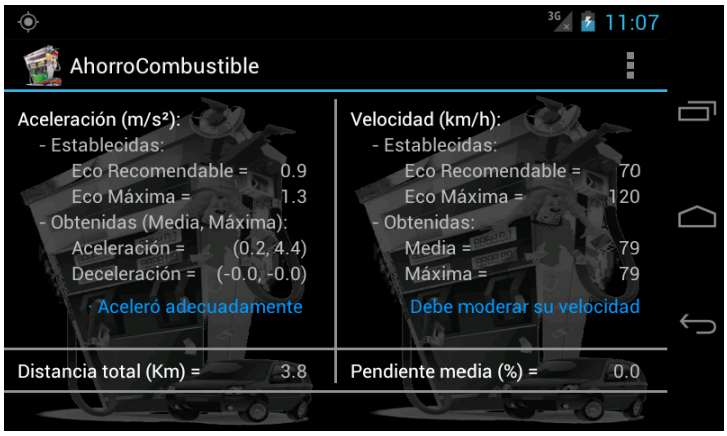


Figura 5.11. Caso de pruebas 3, pantalla resumen

5.1.4 Cuarto caso de pruebas en el emulador

En los casos anteriores hemos probado el cálculo de velocidad de nuestra aplicación, en esta ocasión, vamos a probar la medida de la aceleración. Para ello, vamos a crear una serie de coordenadas encargadas de generar una aceleración constante positiva. Estas serán las condiciones iniciales de la prueba:

- Aceleración constante positiva de 2,2 m/s^2.
- Límites para la aceleración y velocidad por defecto.
- Sin pendiente.

En la **figura 5.12** se muestra el Script generado para este caso de pruebas:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
echo "geo fix 0.0 0.0005"
 sleep 5
 echo "geo fix 0.0 0.0015"
 sleep 5
 echo "geo fix 0.0 0.0030"
 sleep 5
 echo "geo fix 0.0 0.0045"
 sleep 5
 echo "geo fix 0.0 0.0060"
 sleep 5
 echo "geo fix 0.0 0.0075") | telnet localhost 5554
```

Figura 5.12. Script cuarto caso de pruebas en el emulador

El Script generado para este caso de pruebas (**figura 5.12**) se ha creado utilizando la información obtenida al generar los Scripts anteriores. De ellos sabemos que:

- *0.0005º = 55,6 m, donde 55,6 · 5 = 40 Km/h*
- *0.001º = 111,2 m, donde 111,2 · 5 = 80 Km/h*
- *0.0015º = 166,8 m, donde 166,2 · 5 = 120 Km/h*

Podemos aplicar directamente estos datos sobre las coordenadas para ir incrementando la velocidad sin modificar el tiempo entre ellas, que permanecerá fijo en 5 segundos.

A partir de estos datos, en el Script anterior, primero mantenemos la velocidad en 0 Km/h, después la incrementamos a 40 Km/h, a 80 Km/h y por último la mantenemos constante a 120 Km/h. Esto produce una aceleración constante:

aceleración = (velocidad final – velocidad inicial) / tiempo

Por lo tanto, utilizando la velocidad en m/s obtenemos la aceleración:

aceleración = 11,11 / 5 = 2,22 m/s^2

La diferencia entre las velocidades es igual para las tres establecidas y el tiempo entre ellas permanece constante así que la aceleración será constante. Utilizando estas coordenadas las pantallas obtenidas de la aplicación son (**figuras 5.13 y 5.14**):

Figura 5.13. Caso de pruebas 4, pantalla principal 1/2

Figura 5.14. Caso de pruebas 4, pantalla principal 2/2

Estas dos figuras (**5.13** y **5.14**) son pantallas consecutivas mostradas por la aplicación, la **figura 5.14** muestra la pantalla generada una coordenada después de la primera (**5.13**). En ellas podemos ver como la aceleración se mantiene en el valor calculado con las velocidades previstas.

La aplicación, correctamente, muestra un aviso indicando que el límite máximo permitido para la aceleración ha sido superado.

En el caso de las etiquetas sobre la pendiente y el consumo, no aparece ningún valor porque no se le han proporcionado suficientes coordenadas para hacerlo. Recordemos que la aplicación necesita 10 coordenadas para calcular una pendiente.

En la pantalla resumen (**figura 5.15**) aparecerán ahora algunos valores interesantes, esta es dicha pantalla:

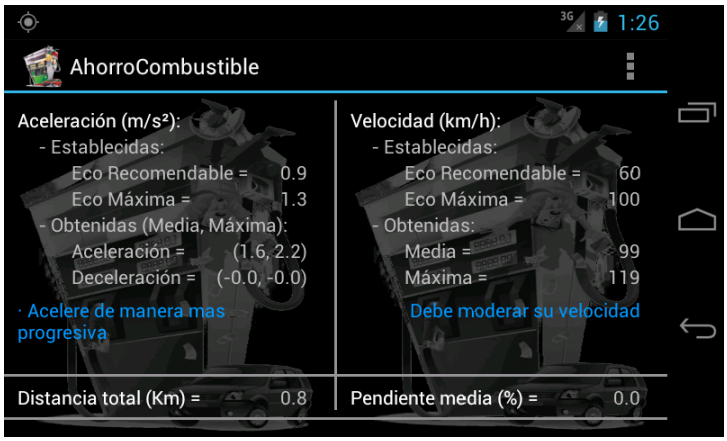


Figura 5.15. Caso de pruebas 4, pantalla resumen

En la sección sobre la aceleración, observamos que el valor de la aceleración media es 1,6 m/s^2 y el valor máximo es 2,2 m/s^2. El valor máximo es el esperado, por el contrario, podemos notar que el valor medio es más bajo de lo previsto, esto se debe, simplemente, a las coordenadas introducidas que mantienen la velocidad inicial y final constantes (aceleración = 0), esto hace bajar la aceleración media.

En el caso de la velocidad, observamos que el valor máximo es 119 Km/h, que se corresponde con lo establecido inicialmente.

5.1.5 Quinto caso de pruebas en el emulador

Vamos a continuar realizando pruebas sobre la aceleración. En esta ocasión, vamos a probar el comportamiento de la aplicación frente a una aceleración negativa (deceleración). Utilizaremos los mismos valores del caso anterior pero haremos el proceso contrario, pasaremos de 120 Km/h a 40 Km/h y mantendremos esta velocidad de manera constante. Estas son las condiciones iniciales del caso de pruebas actual:

- Aceleración constante negativa de -2,2 m/s^2.
- Límites por defecto.
- Sin altitud.

El Script empleado en esta ocasión tiene el aspecto que muestra la **figura 5.16**:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0015"
 sleep 5
 echo "geo fix 0.0 0.0030"
 sleep 5
 echo "geo fix 0.0 0.0045"
 sleep 5
 echo "geo fix 0.0 0.0060"
 sleep 5
 echo "geo fix 0.0 0.0070"
 sleep 5
 echo "geo fix 0.0 0.0075"
 sleep 5
 echo "geo fix 0.0 0.0080"
 sleep 5
 echo "geo fix 0.0 0.0085"
 sleep 5
 echo "geo fix 0.0 0.0090"
 sleep 5
 echo "geo fix 0.0 0.0095"
 sleep 5
 echo "geo fix 0.0 0.0100"
 sleep 5
 echo "geo fix 0.0 0.0105"
 sleep 5
echo "geo fix 0.0 0.0110") | telnet localhost 5554
```

Figura 5.16. Script quinto caso de pruebas en el emulador

Como se ha comentado anteriormente, las coordenadas establecen una velocidad constante al inicio de 120 Km/h, después bajan la velocidad a 80 Km/h y, por último, reducen hasta 40 Km/h, velocidad que permanece constante hasta el final.

Las figuras **5.17** y **5.18** representan dos pantallas consecutivas extraídas de la ejecución de la aplicación mientras se le aplican las coordenadas establecidas en la **figura 5.16** a la misma:

Figura 5.17. Caso de pruebas 5, pantalla principal 1/2

Figura 5.18. Caso de pruebas 5, pantalla principal 2/2

A diferencia del caso de pruebas anterior, en esta ocasión, la aplicación marca como negativa la aceleración, de esta forma, la plataforma entiende que se está realizando una frenada brusca que excede el límite máximo para la aceleración, negativa en este caso. La aplicación lanza mensajes al usuario indicando que este debe realizar las deceleraciones de manera más gradual.

En la pantalla resumen (**figura 5.19**), podemos comprobar que los datos de aceleración de este caso de pruebas se muestran en la parte sobre aceleración negativa (deceleración), como no hay aceleración positiva, los campos correspondientes se encuentran a cero.

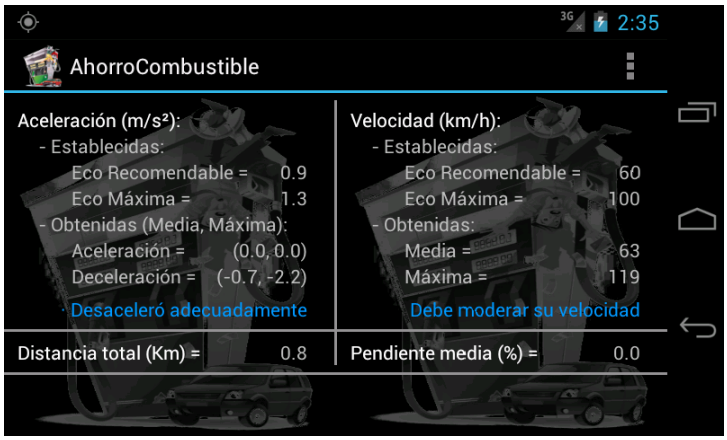


Figura 5.19. Caso de pruebas 5, pantalla resumen

5.1.6 Sexto caso de pruebas en el emulador

Esta vez vamos a establecer un caso de pruebas un poco más complejo, consistirá en pasar de 0 a 120 Km/h mediante una aceleración constante, acto seguido se generará una deceleración más brusca que la aceleración, esta perderá intensidad hasta alcanzar una velocidad constante de 10 Km/h. Esta será la secuencia:

0 Km/h → (+5s) 40 Km/h → (+5s) 80 Km/h → (+5s) 120 Km/h → (+5s) 60 Km/h → (+5s) 10 Km/h

Además de utilizar los datos obtenidos anteriormente, vamos a necesitar calcular la distancia entre coordenadas necesaria para generar una velocidad de 60 Km/h y de 10 Km/h.

- ***60 Km/h*** *= 16,66 m/s → 16,66 · 5 = 83,3 m →*
  distancia (º) = 83,3 · 0.001 / 111,2 = ***0.00075º***
- ***10 Km/h*** *= 2,77 m/s → 2,77 · 5 = 13,85 m →*
  distancia (º) = 13,85 · 0.001 / 111,2 = ***0.000125º***

Los valores para la aceleración serán los mismos que los calculados en el caso de pruebas 4, por tanto, será una aceleración constante de **2,2** m/s^2. Vamos a calcular los valores para la deceleración resultantes de aplicar la reducción de velocidad mostrada anteriormente:

- ***aceleración*** *(120 Km/h a 60 Km/h en 5 s) = (16,66 - 33,36) / 5 =* ***-3,34*** m/s^2
- ***aceleración*** *(60 Km/h a 10 Km/h en 5 s) = (2,77 - 16,66) / 5 =* ***-2,78*** m/s^2

Las condiciones iniciales para este caso de pruebas son:

- Velocidad y aceleración variables.
- Límites para la velocidad por defecto.
- Límites para la aceleración: recomendable = 2,5 m/s^2, máxima = 3.0 m/s^2.
- Sin pendientes.

El Script que contiene las coordenadas que vamos a emplear para generar este caso en el emulador de Android se muestra en la **figura 5.20**:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0"
 sleep 5
 echo "geo fix 0.0 0.0005"
 sleep 5
 echo "geo fix 0.0 0.0015"
 sleep 5
 echo "geo fix 0.0 0.0030"
 sleep 5
 echo "geo fix 0.0 0.00375"
 sleep 5
 echo "geo fix 0.0 0.003875"
 sleep 5
 echo "geo fix 0.0 0.003875"
 sleep 5
 echo "geo fix 0.0 0.003875"
 sleep 5
 echo "geo fix 0.0 0.003875") | telnet localhost 5554
```

Figura 5.20. Script sexto caso de pruebas en el emulador

Como en los casos anteriores, se han establecido al inicio y al final unas coordenadas que mantienen constante la velocidad con el objetivo de observar mejor el caso generado.

La pantalla de configuración inicial para este caso podemos observarla gracias a la **figura 5.21**:

Figura 5.21. Caso de pruebas 6, pantalla de configuración

Una vez nos encontremos en la pantalla principal, arrancamos el Script que va a suministrar a nuestra aplicación las coordenadas. Vamos a mostrar, a continuación, la secuencia de imágenes consecutivas generadas por la aplicación debido a las coordenadas introducidas (**figuras 5.22**, **5.23**, **5.24**, **5,25** y **5.26**):

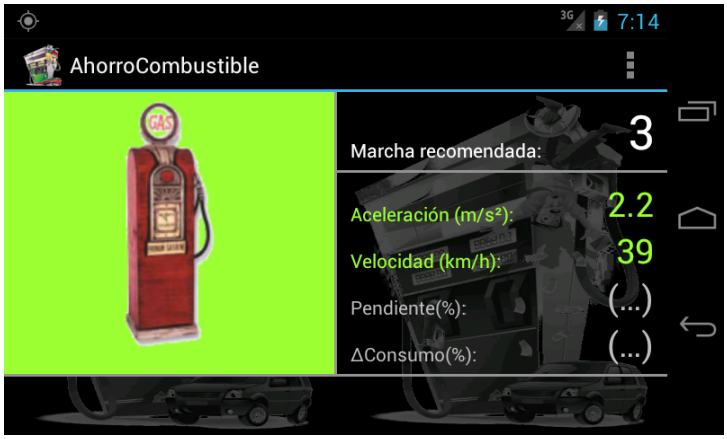


Figura 5.22. Caso de pruebas 6, pantalla principal 1/5

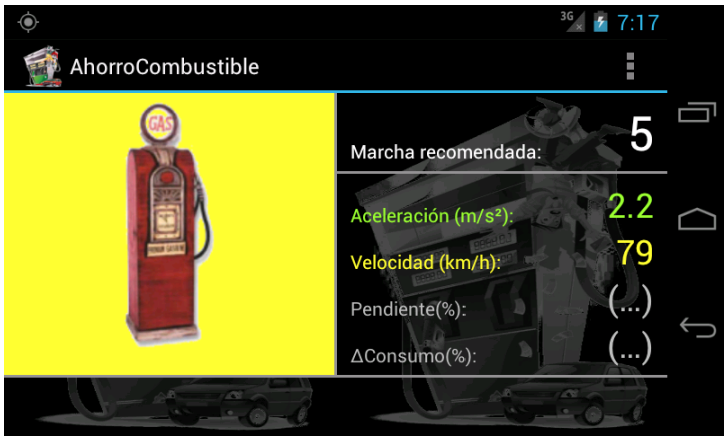


Figura 5.23. Caso de pruebas 6, pantalla principal 2/5

Figura 5.24. Caso de pruebas 6, pantalla principal 3/5

Figura 5.25. Caso de pruebas 6, pantalla principal 4/5

Figura 5.26. Caso de pruebas 6, pantalla principal 5/5

Mediante la batería de figuras mostradas podemos observar perfectamente la evolución de la aplicación en función de las coordenadas recibidas y los límites establecidos.

Durante la aceleración, esta permanece constante y por debajo del límite recomendable (**figuras 5.22**, **5.23** y **5.24**), en esa parte de la prueba, es la velocidad la

que supera progresivamente los límites establecidos y, observamos, también, como la aplicación nos informa paulatinamente de esa situación (**figuras 5.23** y **5.24**).

Durante la deceleración (**figuras 5.25** y **5.26**) se superan los límites establecidos para la aceleración y no se superan los de la velocidad, por tanto, la aplicación nos indica que debemos reducir la velocidad paulatinamente (suavizar la deceleración).

5.1.7 Séptimo caso de pruebas en el emulador

En este séptimo caso de pruebas vamos a comprobar el comportamiento de la aplicación frente a las vías con pendiente. Probaremos que la aplicación calcula correctamente dichas pendientes y que es capaz de ajustar los límites sobre aceleración y velocidad en función de dicha pendiente.

El caso de pruebas que vamos a desarrollar consiste en una batería de coordenadas que junto a una serie de valores de altitud formarán el siguiente escenario:

1. 5 coordenadas que establecen una velocidad constante de 40 Km/h y altitudes para pendiente 0%.
2. 5 coordenadas para fijar la velocidad en 40 Km/h y altitudes para pendiente 5%.
3. 5 coordenadas para establecer una velocidad constante de 80 Km/h y altitudes para pendiente 10%.
4. 5 coordenadas que fijan la velocidad a 80 Km/h y altitudes para una pendiente del 10%.
5. 5 coordenadas para mantener la velocidad en 80 Km/h y altitudes para una pendiente descendente del -20%.
6. 5 coordenadas para mantener la velocidad y la pendiente en los valores del punto 5.

Hay que tener en cuenta que para calcular la pendiente la aplicación hará uso de la altitud media calculada en el punto anterior. Por ejemplo, para calcular la pendiente del caso 3, empleará la altitud media en dicho caso y la altitud media del caso 2.

Vamos a simular una situación en la que primero viajamos a 40 Km/h por una vía sin pendiente, después nos encontramos viajando a la misma velocidad por una pendiente ascendente del 5%. Siguiendo en la vía con la misma pendiente, aumentamos la velocidad a 80 Km/h. Poco después, encontramos que circulando a la misma velocidad la pendiente aumenta al 10%. Por último, mantenemos la velocidad en 80 Km/h pero, ahora, la vía posee una pendiente descendente del 20%.

El objetivo de este caso de pruebas es observar como la aplicación calcula correctamente las pendientes, además, si situamos el límite máximo de velocidad cerca de 80 Km/h cuando circulemos por la pendiente ascendente a esa velocidad la aplicación indicará que hemos rebasado dicho límite, por el contrario, cuando circulemos por la pendiente descendente, a la misma velocidad, la aplicación nos

indicará que nos encontramos dentro de los límites permitidos. Esto es debido al ajuste que hace la plataforma de los límites de aceleración y velocidad en función de la pendiente y la velocidad, como ya comentamos en el apartado correspondiente (**apartado 3.2.4.7**). Además, podremos comprobar que los cálculos que realiza el sistema sobre la variación del consumo se corresponden con los cálculos teóricos.

Vamos ahora a mostrar el Script que contiene las coordenadas encargadas de fijar la velocidad en los valores descritos (**figura 5.27**). En esta ocasión, las coordenadas están separadas entre sí 1 segundo, esto es debido a que necesitamos 5 coordenadas para calcular una altura, separando las coordenadas 5 segundos, como en los casos anteriores, la prueba se prolongaría demasiado tiempo.

Antes de mostrar el contenido del Script, vamos a mostrar la forma en la que hemos calculado la distancia entre las coordenadas para cumplir con las velocidades fijadas. En este caso, tenemos dos velocidades distintas, 40 Km/h y 80 Km/h, como ya hemos utilizado estas velocidades en casos anteriores sólo tenemos que recuperar los datos ya calculados:

- 40 Km/h = 11,11 m/s, la distancia recorrida en 1 segundo son 11,11 m, que traducido a las coordenadas son 0.0001º.
- 80 Km/h = 22,22 m/s, la distancia recorrida en 1 segundo son 22,22 m, traducido a grados para las coordenadas son 0.0002º.

Este es el Script (**figura 5.27**) resultante de aplicar todos los datos descritos:

```
#!/bin/bash
#Script lanza coordenadas gps

cd /home/javier/android-sdk-linux/tools

( echo "geo fix 0.0 0.0"
 sleep 1
 echo "geo fix 0.0 0.0"
 sleep 1
 echo "geo fix 0.0 0.0001"
 sleep 1
 echo "geo fix 0.0 0.0002"
 sleep 1
 echo "geo fix 0.0 0.0003"
 sleep 1
 echo "geo fix 0.0 0.0004"
 sleep 1
 echo "geo fix 0.0 0.0005"
 sleep 1
 echo "geo fix 0.0 0.0006"
 sleep 1
 echo "geo fix 0.0 0.0007"
 sleep 1
 echo "geo fix 0.0 0.0008"
 sleep 1
 echo "geo fix 0.0 0.0009"
 sleep 1
 echo "geo fix 0.0 0.0010"
 sleep 1
```

```
echo "geo fix 0.0 0.0012"
sleep 1
echo "geo fix 0.0 0.0014"
sleep 1
echo "geo fix 0.0 0.0016"
sleep 1
echo "geo fix 0.0 0.0018"
sleep 1
echo "geo fix 0.0 0.0020"
sleep 1
echo "geo fix 0.0 0.0022"
sleep 1
echo "geo fix 0.0 0.0024"
sleep 1
echo "geo fix 0.0 0.0026"
sleep 1
echo "geo fix 0.0 0.0028"
sleep 1
echo "geo fix 0.0 0.0030"
sleep 1
echo "geo fix 0.0 0.0032"
sleep 1
echo "geo fix 0.0 0.0034"
sleep 1
echo "geo fix 0.0 0.0036"
sleep 1
echo "geo fix 0.0 0.0038"
sleep 1
echo "geo fix 0.0 0.0040"
sleep 1
echo "geo fix 0.0 0.0042"
sleep 1
echo "geo fix 0.0 0.0044"
sleep 1
echo "geo fix 0.0 0.0046"
sleep 1
echo "geo fix 0.0 0.0048"
sleep 1
echo "geo fix 0.0 0.0050") | telnet localhost 5554
```

Figura 5.27. Script séptimo caso de pruebas en el emulador

Una vez resuelto el caso de la velocidad, ahora vamos a encargarnos de calcular las altitudes necesarias que introduciremos a la aplicación para que muestre las pendientes deseadas. Para ello, sabemos que:

Pendiente (%) = Diferencia de altitud · 100 / Distancia total

La distancia total es la suma de las distancias recorridas durante diez coordenadas, ya que necesitamos cinco coordenadas para cada valor de altitud y necesitamos dos valores de altitud para la pendiente. Teniendo en cuenta nuestras velocidades, el valor de la distancia cada 5 coordenadas para las dos velocidades es:

- 5 coordenadas a 40 Km/h son 55,55 metros.
- 5 coordenadas a 80 Km/h son 111,11 metros.

Despejando de la ecuación de la pendiente el valor de la diferencia de altitud:

Δaltitud = pendiente (%) · distancia total / 100

Vamos ahora a calcular la diferencia de altitud entre dos valores de esta consecutivos para lograr las pendientes que necesitamos en cada uno de los casos propuestos para la prueba:

- 5 coordenadas a 40 Km/h seguidas de otras 5 coordenadas a la misma velocidad que dan como resultado una pendiente del 5%:
  Δaltitud = 5 · (55,55 + 55,55) / 100 = ***5,55 m***

- 5 coordenadas a 40 Km/h seguidas de otras 5 coordenadas a 80 Km/h, la pendiente en este caso es del 10%:
  Δaltitud = 10 · (55,55 + 111,11) / 100 = ***16,66 m***

- 5 coordenadas a 80 Km/h seguidas de otras 5 coordenadas a la misma velocidad, la pendiente es también del 10%:
  Δaltitud = 10 · (111,11 + 111,11) / 100 = ***22,22 m***

- 5 coordenadas a 80 Km/h seguidas de otras 5 coordenadas a la misma velocidad, la pendiente será del -20%:
  Δaltitud = -20 · (111,11 + 111,11) / 100 = ***-44,44 m***

- 5 coordenadas a 80 Km/h seguidas de otras 5 coordenadas a la misma velocidad, la pendiente será del -20%:
  Δaltitud = -20 · (111,11 + 111,11) / 100 = ***-44,44 m***

En los cálculos descritos no se han citado las 5 coordenadas iniciales que no poseen pendiente alguna.

Para introducir los valores de altitud no podemos utilizar directamente el emulador de Android ya que este no está preparado para tal asunto. Debemos, por tanto, introducir directamente los valores de altitud en el código de nuestra aplicación. Para tal fin, dentro de la subclase **MyLocationListener** se han definido dos atributos (**código fuente 36**):

```
private int contadorArrayALturas;
private int[] alturasPredef = {
          600, 600, 600, 600, 600, 600, 600,
          606, 606, 606, 606, 606,
          623, 623, 623, 623, 623,
          644, 644, 644, 644, 644,
          600, 600, 600, 600, 600,
          556, 556, 556, 556, 556};
```

Código Fuente 36. Atributos de la clase MyLocationListener para la séptima prueba

Ya que conocemos el número de coordenadas que vamos a introducir para la prueba, podemos crear un **array** del mismo tamaño para las altitudes en cada una de las

coordenadas. En este **array** llamado "**alturasPredef**" se han incluido, además, las dos coordenadas iniciales que realmente no tienen efecto sobre el resto.

Para crear los valores de altitud se ha establecido como altitud inicial 600 metros, aunque podría haber sido cualquier otra. Siguiendo los valores calculados anteriormente sobre la diferencia entre alturas se ha completado el resto del **array**, teniendo en cuenta que en nuestra aplicación la altura es un número entero y por ello se han aproximado los valores al entero más próximo.

El atributo "**contadorArrayAlturas**" se encarga de almacenar el índice del **array** para que podamos irlo recorriendo de forma secuencial cada vez que llegue una coordenada nueva.

Además de crear los atributos anteriores, que serán borrados después de la prueba, se ha establecido el valor inicial para el atributo "**altitud**" en 600 metros.

Estableciendo este valor inicial conseguimos que en la 7ª coordenada, que sería la 5ª real para la aplicación (ya que sabemos que esta necesita dos coordenadas para iniciar correctamente los valores de velocidad, aceleración y distancia), ya podamos calcular la primera pendiente, en este caso, después de las siete primeras coordenadas introducidas por el Script la aplicación mostrará el valor de la pendiente en 0. En el caso de mantener el valor de este atributo a 0 no se indicaría valor alguno de pendiente hasta la 12ª coordenada.

Se ha creado un método que se encarga de establecer los valores adecuados de altitud para cada coordenada, este es dicho método (**código fuente 37**):

```
public float generarAltitud() {

    float altitudGenerada = 0;

    altitudGenerada = (float)alturasPredef[contadorArrayALturas];
    contadorArrayALturas++;

    return altitudGenerada;

}
```

Código Fuente 37. Método generarAltitud() de la clase MyLocationListener

Como se puede observar en el **código fuente 37**, este método se encarga de recorrer el **array** de altitudes y devolver el valor correspondiente en cada caso.

Para introducir los valores de altitud en cada uno de los objetos **Location** incluiremos el siguiente código (**código fuente 38**) al inicio del método ***onLocationChanged*()**:

```
loc.setAltitude(generarAltitud());
```

Código Fuente 38. Llamada método Location.setAltitude() desde onLocationChanged()

Antes de lanzar el caso de prueba, vamos a estimar el valor del factor pendiente en los casos más relevantes para poder observar de esta forma si el cálculo de este y la adaptación de los límites en función del mismo se hacen correctamente, utilizaremos para los cálculos el algoritmo descrito en el **apartado 3.2.4.7** del presente proyecto:

- Factor pendiente resultante a 40 Km/h con pendiente del 5%:

  E_c (40 Km/h) = 7405,92 J
  Relación energía pendiente = 1,09
  *"**factorPendiente**" = 0,91*

  Es decir, el consumo se incrementará un 9%, además, los límites de velocidad y aceleración serán un 9% más restrictivos.

- Factor pendiente resultante a 80 Km/h con pendiente del 10%:

  E_c (80 Km/h) = 29.623,7 J
  Relación energía pendiente = 1,09
  "**factorPendiente**" = 0,91

  Es decir, el consumo se incrementará un 9%, además, los límites de velocidad y aceleración serán un 9% más restrictivos.

- Factor pendiente resultante a 80 Km/h con pendiente del -20%:

  E_c (40 Km/h) = 7405,92 J
  Relación energía pendiente = 0,82
  "**factorPendiente**" = 1,18

  Es decir, el consumo se reducirá un 18%, además, los límites de velocidad y aceleración serán un 18% más permisivos.

Con todos los datos que debe mostrar la aplicación previamente calculados, ya podemos lanzar la aplicación, la pantalla inicial de configuración tendrá el siguiente aspecto (**figura 5.28**):

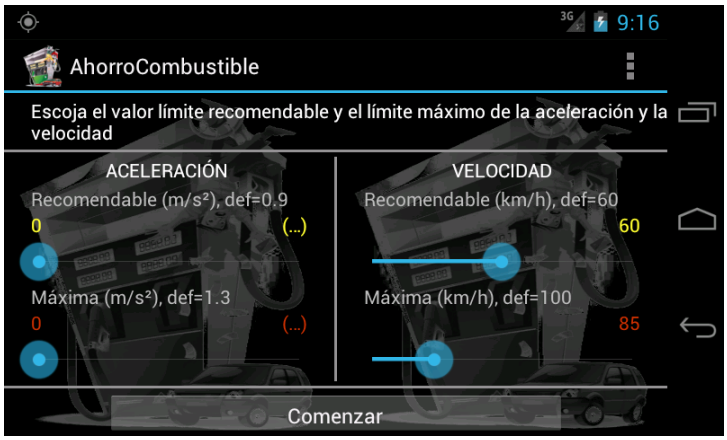


Figura 5.28. Caso de pruebas 7, pantalla de configuración

En la **figura 5.28** se observa que se han establecido los límites por defecto para la aceleración, respecto la velocidad, se ha escogido el valor 60 Km/h para el límite recomendable y 85 Km/h para el límite máximo, este segundo valor nos va a permitir comprobar fácilmente la influencia del "**factorPendiente**" en los límites establecidos.

A continuación, vamos a mostrar la batería de imágenes obtenidas de aplicar todo lo comentado anteriormente sobre nuestra aplicación (**figuras 5.29, 5.30, 5.31, 5.32, 5.33 y 5.34**):

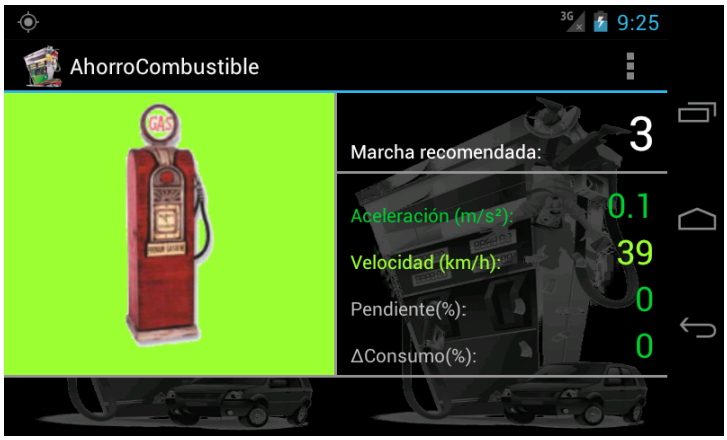


Figura 5.29. Caso de pruebas 7, pantalla principal 1/6

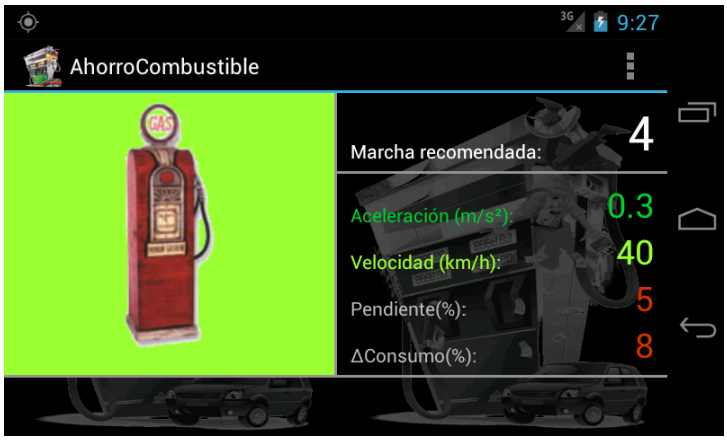


Figura 5.30. Caso de pruebas 7, pantalla principal 2/6

Figura 5.31. Caso de pruebas 7, pantalla principal 3/6

Figura 5.32. Caso de pruebas 7, pantalla principal 4/6

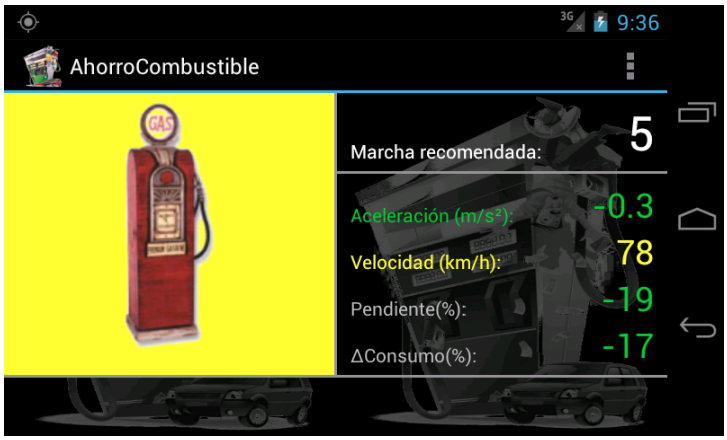


Figura 5.33. Caso de pruebas 7, pantalla principal 5/6

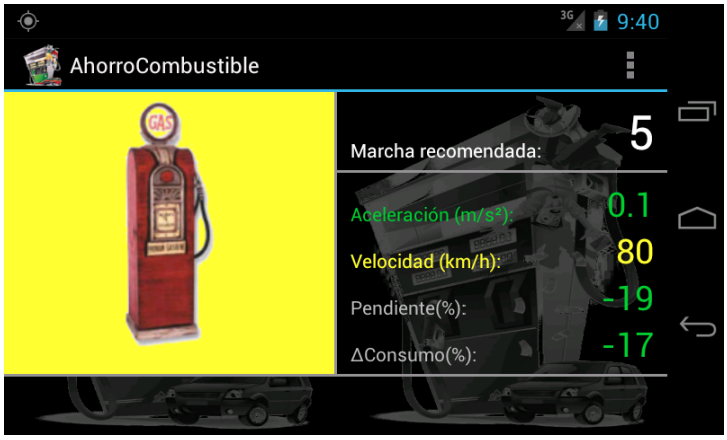


Figura 5.34. Caso de pruebas 7, pantalla principal 6/6

Como se puede observar en la batería de imágenes (**figuras 5.29, 5.30, 5.31, 5.32, 5.33 y 5.34**) tanto la velocidad como los valores de la pendiente se corresponden con los valores esperados en cada caso. Con una pequeña diferencia debido a la falta de precisión con los decimales durante los cálculos.

En la **figura 5.30** se muestra una variación del consumo del 8%, debida a circular a 40 Km/h por una pendiente del 5%. En los cálculos previos este valor era del 9% por lo que podemos considerar correcto este valor.

En la **figuras 5.31** y **5.32** también se muestran los valores estimados teóricamente para la variación del consumo. En estas figuras podemos observar el cambio en el valor del límite establecido para la velocidad máxima. Según los cálculos previos, los límites serán un **9%** más restrictivos, debido a esto, el límite máximo para la velocidad pasará de ser **85 Km/h** a ser **77 Km/h**. La aplicación muestra, correctamente, que estamos superando el límite máximo de velocidad en esta situación.

En las **figuras 5.33** y **5.34** podemos observar la situación contraria a la descrita en el párrafo anterior. La pendiente es descendente en este caso y, por ello, el consumo del vehículo se reducirá un 18%, como se calculó previamente. En esta situación, la

aplicación indica que hemos superado el límite recomendable para la velocidad, por el contrario, no hemos superado el límite intermedio situado entre el límite recomendable y el límite máximo (para la configuración establecida inicialmente este límite valdría **72 Km/h**) que en este caso, debido a la pendiente, valdrá **84 Km/h**, mientras que el límite máximo valdrá **100 Km/h**. El límite que la aplicación indica que hemos superado es el límite recomendable para la velocidad, que se configuró como 60 Km/h, el cual se ha adaptado y en estas circunstancias vale **70 Km/h**.

5.2 Pruebas de precisión y situación real en un vehículo

En este apartado vamos a realizar una prueba de nuestra aplicación en una situación real. Pondremos en marcha nuestra aplicación dentro de un vehículo turismo y realizaremos un viaje que nos servirá para efectuar algunas comprobaciones. Los objetivos de la prueba son:

- Comprobar la precisión del marcador de velocidad de la aplicación respecto al velocímetro del vehículo.
- Comprobar la distancia total señalada por la aplicación respecto al cuentakilómetros del vehículo.
- Comprobar las latencias de la aplicación.
- Comprobar que la aplicación muestra datos que se corresponden con la situación real del vehículo respecto a las aceleraciones y la pendiente.
- Verificar la reproducción de los mensajes sonoros correspondientes en cada situación.

Para la realización de esta prueba utilizaremos un vehículo Volkswagen Passat con motor de inyección de gasolina de 2000 centímetros cúbicos.

Veamos ahora los resultados obtenidos en cada uno de los apartados de la presente prueba.

5.2.1 Precisión del marcador de velocidad

Durante el viaje de prueba se tomaron varias muestras de este dato de forma simultánea en el marcador de la aplicación y en el velocímetro del vehículo. Para tomar estas muestras se estableció una velocidad constante en el vehículo, utilizando el velocímetro del mismo. En la **tabla 5.1** se muestran los resultados obtenidos.

Velocímetro Coche (Km/h)	Marcador Aplicación (Km/h)
50	43
90	84
100	93
120	114

Tabla 5.1. Diferencias velocímetros

En la **tabla 5.1** vemos que en todos los casos el marcador de nuestra aplicación muestra una velocidad inferior que la indicada por el velocímetro del vehículo. Esta diferencia se debe a la precisión de ambos marcadores.

Sabemos que los velocímetros de los vehículos poseen un error medio del 5% respecto a la velocidad real. Además, es conocido que dichos marcadores, por motivos de seguridad, están diseñados para marcar siempre una velocidad superior a la real, teniendo en cuenta el error descrito. **[19]**

De los datos obtenidos de la prueba podemos extraer que la diferencia resultante entre ambos valores permanece constante en **6-7 Km/h**. Teniendo en cuenta todo lo anterior, podemos concluir que la aplicación marca correctamente el valor de la velocidad real del vehículo, incluso con mayor precisión que el propio velocímetro del vehículo.

5.2.2 Precisión distancia total

Para comprobar este campo, mostrado por nuestra aplicación en la pantalla resumen, utilizaremos el cuentakilómetros parcial de nuestro vehículo. El proceso comenzará poniendo a cero dicho marcador, al final del recorrido compararemos el valor mostrado por el cuentakilómetros del coche y el valor correspondiente mostrado en la pantalla resumen de nuestra aplicación.

Durante el viaje de pruebas, según el cuentakilómetros del coche, se han recorrido en total **24,7 Km**. Según el indicador de nuestra aplicación, la distancia total recorrida es de **24,3 Km**.

A la vista de los resultados obtenidos, podemos concluir que el cálculo de la distancia total realizado por la aplicación es correcto.

5.2.3 Latencias de la aplicación

Para comprobar las latencias mostradas por la aplicación se han llevado a cabo cambios en la situación del vehículo durante el viaje, de esta forma, podemos observar el tiempo que tarda la aplicación en reaccionar a los cambios producidos en la aceleración y velocidad del vehículo.

Estos cambios consistirán en:

- Circulando a una velocidad constante, soltar el acelerador del vehículo y dejar rodar el mismo por su propia inercia.
- Circulando a una velocidad constante, soltar el acelerador y pisar el freno en diferentes intensidades.
- Circulando a una velocidad constante, realizar más presión sobre el pedal del acelerador para aumentar la velocidad.

Llevando a cabo dichas pruebas se comprobó que la aplicación tardaba **1 coordenada** (separadas estas entre sí 1 segundo en condiciones normales) en adaptarse a la nueva situación del vehículo. Se observó que la aplicación mostraba la nueva situación y también la transición hasta dicha situación.

Por otro lado, referente al marcador de la pendiente de la vía, se observó que la aplicación tarda, aproximadamente, 5 segundos en mostrar la nueva pendiente de la vía por la que circulemos. Como sabemos, esta latencia es debida al número de coordenadas que necesita nuestra aplicación para realizar el cálculo de la pendiente.

5.2.4 Datos referentes a la aceleración y la pendiente

En esta ocasión, no podemos realizar comprobaciones de precisión respecto a estos datos sin diseñar unas pruebas mucho más exhaustivas (no procedentes para el objetivo final de nuestra aplicación). De todas formas, conocemos el correcto funcionamiento de la aplicación respecto al cálculo de dichos datos mediante las pruebas realizadas en el emulador.

En esta parte de la prueba podemos verificar que la aplicación siempre mostró el signo (signo positivo para aceleración y pendiente ascendente, signo negativo para deceleración y pendiente descendente) adecuado en cada uno de los datos y que estos variaban correctamente en función de la intensidad de la aceleración y de la pendiente.

5.2.5 Verificación de mensajes sonoros

Durante la realización de la presente prueba se comprobó que la aplicación reproduce los mensajes apropiados a cada situación mostrada en la pantalla, incluyendo el mensaje sobre detener el motor en paradas prolongadas. Además, se comprobó que estos mensajes no se solapan entre sí, estos se reproducen espaciados en el tiempo.

5.3 Fase de pruebas de variación del consumo en circuito real

El objetivo de este caso de pruebas es cuantificar la diferencia real en el consumo entre dos situaciones distintas, cuando circulamos conduciendo de la manera habitual y cuando lo hacemos circulando bajo las recomendaciones de nuestra aplicación.

Para realizar esta prueba es necesario, en ambos casos: utilizar el mismo coche, el mismo conductor, la misma marca, el mismo tipo de gasolina y, sobre todo, realizar el mismo recorrido en circunstancias semejantes.

La prueba conllevará los siguientes pasos:

1. Medimos la gasolina del vehículo.
2. Realizamos el recorrido establecido conduciendo normalmente.
3. Medimos la gasolina gastada y comprobamos los kilómetros recorridos para calcular el consumo.
4. Realizamos el recorrido establecido, de nuevo, esta vez utilizando nuestra aplicación.
5. Medimos la gasolina gastada y los kilómetros recorridos durante esta segunda parte.

El vehículo utilizado para tal fin es un Volkswagen Passat con motor de inyección de gasolina de 2000 centímetros cúbicos. El combustible empleado es gasolina EuroSuper 95 de la marca BP.

Por su parte, el recorrido tiene una longitud total aproximada de 147 Km, la mayor parte de este recorrido son tramos de autovía con pendientes relativamente pequeñas, menores de un 5%. Los límites de velocidad oscilan en la mayoría del recorrido entre 70 y 120 Km/h, con algunos tramos urbanos de límite 50 Km/h. El recorrido también incluye algunos tramos de circulación por ciudad que implican semáforos, glorietas, pasos de peatones elevados y demás obstáculos.

Cabe destacar que dicho recorrido se realizará en aquellos momentos en los que la circulación sea más fluida, de esta forma evitaremos atascos que no nos van a aportar ningún tipo de información sobre el consumo en los distintos casos previstos.

Para medir la gasolina empleada en cada uno de los dos casos utilizaremos el medidor que ofrece el surtidor de la gasolinera (en los dos casos emplearemos el mismo surtidor de la misma gasolinera). Para llevar a cabo las mediciones el proceso será el siguiente:

1. Llenamos el depósito del vehículo completamente.
2. Realizamos el recorrido una vez.

3. Volvemos a llenar el depósito, el surtidor nos indicará el volumen de gasolina que entra en el depósito. Esta será la gasolina que hemos gastado en este primer viaje.
4. Realizamos el recorrido por segunda vez.
5. Llenamos de nuevo el depósito para averiguar la gasolina empleada esta segunda vez.

Comprobando el indicador de distancia del vehículo y el marcador del surtidor podremos calcular fácilmente la diferencia entre el consumo cuando utilizamos la aplicación y cuando no lo hacemos.

Se realizará la primera parte de la prueba teniendo en cuenta las consideraciones anteriores. Para esta primera parte de la prueba conduciremos nuestro vehículo dejándonos llevar por la tendencia del resto del tráfico, es decir, mantendremos una velocidad y una aceleración acorde al resto de vehículos.

Esta manera de conducir implica aceleraciones y deceleraciones pronunciadas cuando conducimos por ciudad y nos encontramos con obstáculos, como son semáforos, glorietas, señales de stop, etc. Durante las aceleraciones, se cambiará de marcha a unas rpm mayores de las aconsejadas por el IDAE para no perder capacidad de aceleración.

Referente a la velocidad, en carretera circularemos siempre a la velocidad máxima permitida en cada vía, nunca excederemos los límites en más de un 5%, evitando de esta manera poder ser sancionados por exceso de velocidad.

En estas condiciones se realizó dicha prueba, empezaremos a comentar los resultados obtenidos con la distancia total recorrida y los litros consumidos en esa distancia:

*Se recorrieron en total **148,2** Km y se consumieron **10,62** litros de combustible.*

Teniendo en cuenta los datos anteriores, el consumo medio cada 100 Km es de:

*Consumo (litros/100 Km) = 10,62 · 100 / 148,2 = **7,16 litros/100 Km***

Para llevar a cabo la segunda parte de la prueba, consistente en circular bajo las indicaciones de la aplicación, se han tomado algunas decisiones previas a su realización. La primera de ellas es dejar la configuración por defecto como configuración inicial. Siempre se circulará por debajo del límite máximo de velocidad y aceleración. Por otro lado, el límite recomendable sólo será superado en aquellas condiciones que impliquen un problema de seguridad en caso de no hacerlo. Esta situación se puede dar en carriles de aceleración y en autovías con límite de velocidad de 120 Km/h, donde circular por debajo de 80-90 Km/h supone riesgo de alcances.

Durante el recorrido, siempre se respetarán los límites de velocidad de la vía, en ningún caso se sobrepasará el límite de 100 Km/h, siendo la velocidad máxima elegida para la prueba durante el viaje de 90 Km/h, en aquellas vías que permitan dicha

velocidad. Se respetarán los consejos de la conducción eficiente referentes a aceleraciones, deceleraciones, velocidad, cambios de marchas y circulación por vías con pendiente, entre otras.

Teniendo en cuenta todas las consideraciones anteriores, después de realizar el recorrido mencionado, el resultado es:

*Se recorrieron en total **147** Km y se han empleado **7,63** litros de gasolina.*

A partir del resultado anterior, podemos concluir que el consumo medio en esta segunda parte de la prueba es de:

*Consumo (litros/100 Km) = 7,63 · 100 / 147 = 5,19 **litros/100 Km***

Se observa una pequeña diferencia en la distancia total recorrida, esto se debe a pequeñas diferencias durante el circuito (por ejemplo, cambios de carril). Esta diferencia es irrelevante ya que en el cálculo del consumo medio cada 100 Km se tiene en consideración la distancia exacta recorrida en cada caso.

En resumen, los resultados obtenidos en ambos casos, expresando el consumo en litros cada 100 Km son:

- *Primera parte, **conducción habitual = 7,16 litros cada 100 Km***
- *Segunda parte, **conducción eficiente = 5,19 litros cada 100 Km***

Como se puede observar en los resultados obtenidos, existe una gran diferencia de consumo en ambos casos. Teniendo en cuenta que el precio de la gasolina se mantiene actualmente cerca de los **1,43 €/litro**, este resultado implica un ahorro de **2,82€** de combustible cada 100 Km.

Si realizamos al año unos **20.000 Km** con nuestro vehículo, conducir de manera eficiente supondría un ahorro aproximado de **564 €** anuales. Sin duda es un ahorro muy significativo.

5.4 Evaluación de requisitos previstos

En este apartado vamos a evaluar cada uno de los requisitos que debe cumplir nuestra aplicación. Iremos recorriendo, uno a uno, la lista de requisitos descrita en el **apartado 3.1.3** del presente proyecto comprobando que la aplicación desarrollada satisface dichos requisitos.

- **RFUN-01, la aplicación debe poder ser configurada por el usuario**: En el **apartado 5.1** se han mostrado varios ejemplos que personalizan la configuración inicial de la aplicación y se ha demostrado que la aplicación se adapta a cualquier configuración.

- **RFUN-02, mostrar información del viaje en tiempo real**: Como el requisito anterior, en el **apartado 5.1** queda demostrado que la aplicación muestra información del viaje en tiempo real, cada vez que llega una coordenada se actualiza la información mostrada en la interfaz y así podemos comprobar que la información mostrada se corresponde con la situación generada por nosotros para realizar la prueba.

- **RFUN-03, enseñar al usuario las claves de la conducción eficiente en tiempo real**: De nuevo, las simulaciones realizadas en el **apartado 5.1** nos sirven para demostrar que la aplicación emite información sobre conducción eficiente mientras corre.

- **RFUN-04, los márgenes se deben adaptar a la pendiente de la carretera**: En el séptimo caso de pruebas, contenido en el **apartado 5.1.7**, se prueba esta capacidad de la aplicación para adaptar los límites sobre aceleración y velocidad en función de la pendiente.

- **RFUN-05, se mostrará un resumen del viaje con los datos más relevantes**: La actividad **ahorroResumen** se encarga de mostrar dicho resumen del viaje, esta pantalla contiene información sobre la aceleración y la velocidad media, la distancia total recorrida y la pendiente media de la vía.

- **RAPA-01, la vista se debe adaptar a la posición del dispositivo, ya sea horizontal o vertical**: Gracias al emulador de Android hemos podido comprobar que la aplicación se adapta a la posición del dispositivo, presionando en el emulador "*ctrl+F12*" se simula el cambio de orientación del dispositivo. A lo largo del **apartado 5.1** hemos visto pantallas de la aplicación en posición horizontal y también en posición vertical.

- **RAPA-02, será compatible con distintos tamaños de pantalla y diferentes densidades de estas**: Podemos simular que nuestra aplicación corre en distintos terminales con diferentes pantallas utilizando el emulador de Android. A continuación veremos algunos ejemplos de cómo se muestra nuestra aplicación en distintos tipos de pantallas. Hasta ahora, todos los ejemplos mostrados de nuestra aplicación se correspondían con una pantalla de tamaño 800x480 píxeles con una densidad de 240 píxeles por pulgada (ppp). En las figuras **5.35**, **5.36** y **5.37** podemos ver la pantalla principal de la aplicación mostrada por dispositivos cuyas características de pantalla son diferentes.

Figura 5.35. Pantalla principal, tamaño = 480x320, densidad = 160

Figura 5.36. Pantalla principal, tamaño = 400x240, densidad = 120

Figura 5.37. Pantalla principal, tamaño = 1280x800, densidad = 320

- **RFAU-01, la interfaz debe ser sencilla e intuitiva**: Se ha creado un grupo heterogéneo de 10 personas para que pruebe la aplicación. Este grupo lo componen personas cuyas edades oscilan entre los 18 y los 48 años, en él encontramos: 3 ingenieros, dos financieros, un enfermero, un estudiante de farmacia, un estudiante de derecho, un camarero y un agricultor.

Cuando se les presentó la aplicación, todos ellos fueron capaces de manejarla sin ningún esfuerzo. Las pantallas de ayuda les fueron muy útiles para aprender a moverse por la aplicación.

- **RFAU-02, el uso de la aplicación no debe requerir conocimientos previos**: Gracias al mismo grupo de personas utilizado en el requisito anterior pudimos comprobar si la aplicación cumple con este requisito. La mayoría de los miembros del grupo coincide en apuntar a la configuración de los límites sobre aceleración como la parte menos intuitiva de la aplicación. Todas aquellas personas que no están familiarizadas con el mundo del motor desconocen a priori que valores de la aceleración se consideran normales, insuficientes o excesivos.

  Para solventar, en la medida de lo posible, dicha situación se ha incluido más información sobre este campo en la ventana de ayuda de la pantalla de configuración.

- **RFAU-03, la aplicación no requerirá un gran nivel de atención del usuario**: Utilizando de nuevo a nuestro grupo de personas heterogéneo conseguimos una evaluación de este requisito. Aunque al principio la mayoría sentía la tentación de mirar constantemente la pantalla y, a su vez, eran sorprendidos por los mensajes de voz, en poco tiempo se acostumbraron a la información auditiva ofrecida por la aplicación y comprobaron que gracias a esta casi nunca les era necesario fijarse en la pantalla para seguir las indicaciones de la aplicación.

- **RFAU-04, el ahorro de combustible producido provocará que el usuario quiera seguir utilizando la aplicación**: Este requisito quedó demostrado en el **apartado 5.3** de este capítulo, donde se realizaron pruebas de diferencia de consumo entre circular siguiendo las instrucciones de la aplicación o no hacerlo.

- **RFAU-05, la apariencia será sencilla para no incrementar el consumo de batería**: En las interfaces resultantes del desarrollo de la aplicación no se encuentran gráficos complejos que se vayan modificando dinámicamente, las interfaces están compuestas mayoritariamente por etiquetas e imágenes estáticas.

- **RFAU-06, la interfaz será auto-explicativa**: Todos los elementos que componen los interfaces de usuario se encuentran explicados en la propia interfaz y también en la ventana de ayuda asociada a cada pantalla de la aplicación.

- **RFAU-07, mensajes cortos, precisos y fácilmente comprensibles**: Para la elaboración de los mensajes se ha buscado indicar la máxima información posible en el menor espacio posible. Así, tanto los mensajes escritos como los mensajes de audio se han formado con 4 palabras o menos, donde se indica al usuario la acción concreta a realizar utilizando palabras que comprende cualquier persona que posea el permiso de conducir.

- **RFAU-08, mensajes de audio suficientemente espaciados en el tiempo y mostrar en la pantalla sólo la información necesaria**: En la pantalla de nuestra aplicación sólo se muestran aquellas etiquetas de información que el usuario necesita para ahorrar combustible, todas ellas sirven al usuario para poder actuar sobre una situación concreta que esté ocurriendo en ese instante.

  Por otro lado, gracias a los atributos creados especialmente para espaciar los mensajes de audio se ha logrado impedir que estos se solapen o que constantemente se emita información auditiva que sature al usuario.

- **RDES-01, procesará coordenadas recibidas con una frecuencia máxima de 1 coordenada por segundo**: A lo largo del **apartado 5.1** hemos visto como la aplicación es capaz de procesar coordenadas recibidas en distintos intervalos de tiempo. En el **apartado 5.1.7** comprobamos que la aplicación procesa correctamente coordenadas separadas entre sí 1 segundo.

- **RDES-02, la latencia ha de ser la mínima posible, lo deseable es que se encuentre entre 1 y 2 segundos**: De igual forma que en el requisito anterior, sabemos que la latencia introducida por nuestra aplicación es menor a 1 segundo, conocemos este dato ya que la aplicación es capaz de procesar y mostrar los resultados de todas las coordenadas recibidas cuando estas llegan separadas entre sí 1 segundo. Por otro lado, a lo que tarda nuestra aplicación en procesar las coordenadas hay que añadir lo que tarda el GPS en generar dichas coordenadas, sabemos que esta operación también ocupa un tiempo inferior a un segundo.

- **RDES-03, indicaciones sonoras de la aplicación**: Este requisito fue evaluado positivamente gracias a la prueba realizada en el **apartado 5.2** donde se comprobó el funcionamiento general de la aplicación en una situación real.

- **RDES-04, error tolerable en la localización del dispositivo menor de 25 metros**: Dentro del código fuente de nuestra aplicación (**apartado 3.2.4.5**) se ha introducido una cláusula que evita que coordenadas con una precisión peor a 25 metros sean procesadas por nuestra aplicación, estas serán descartadas.

  Por otro lado, en la prueba descrita en el **apartado 5.2** se ha comprobado que los valores calculados por la aplicación se corresponden a los valores mostrados por el vehículo (con una pequeña diferencia debido a la precisión de los instrumentos de medida).

- **RDES-05, utilizar varias muestras para calcular la altura respecto al nivel del mar**: En el **apartado 3.2.4.6** de la presente memoria se detalla el algoritmo de procesamiento de los datos de altitud con objeto de mejorar la precisión de estos. Por otro lado, como se detalló en el **apartado 5.1.7** se ha comprobado el correcto funcionamiento de dicho algoritmo. Además, durante la prueba

realizada en el **apartado 5.2** comprobamos que los datos ofrecidos por la aplicación sobre la pendiente se corresponden con la realidad.

- **RDES-06, la aplicación estará preparada para recibir coordenadas de manera no uniforme respecto al tiempo**: Como hemos visto en el **apartado 3.2.4.5** la aplicación asume que las coordenadas no llegan en instantes de tiempo concreto sino que más bien el tiempo entre ellas depende de muchos factores. La aplicación, por tanto, realiza los cálculos necesarios teniendo en cuenta el tiempo que ha pasado entre la última coordenada recibida y la anterior. Teniendo en cuenta, además, que la aplicación ha sido probada en una situación real (**apartado 5.2**), podemos concluir entonces que la aplicación responde bien aunque las coordenadas no lleguen igualmente espaciadas en el tiempo.

- **RDES-07, descartar coordenadas antiguas**: Como en el requisito anterior, en el **apartado 3.2.4.5** se describe como descarta nuestra aplicación aquellas coordenadas que son más antiguas que la última coordenada recibida y procesada.

- **ROPA-01, el producto se utilizará dentro de un vehículo, este ha de encontrarse al aire libre fuera de una zona de sombra producida por un edificio**: Para las pruebas de los **apartados 5.2** Y **5.3** del presente capítulo el sistema se probó en un dispositivo que se encontraba ubicado dentro del vehículo en el que se han realizado las pruebas.

- **ROPA-02, la aplicación será diseñada para usarse en vehículos turismos**: La aplicación ha sido diseñada en base a las recomendaciones del IDAE sobre conducción eficiente para vehículos turismos, estas recomendaciones pueden no ser útiles en el caso de otro tipo de vehículos.

- **ROPA-03, debe ser compatible con el mayor número posible de versiones de Android disponibles hasta la fecha y posteriores**: La aplicación ha sido probada en el emulador, simulando en este todos los API desde el 1 hasta el 15, ambos incluidos.

- **ROPA-04, permitirá atender llamadas o mensajes de cualquier tipo entrantes al dispositivo durante la ejecución de la aplicación, esta retomará posteriormente su funcionamiento normal**: Gracias al emulador se ha podido simular aquellas situaciones en las que se produce una llamada entrante o llega un mensaje durante la ejecución de la aplicación. El resultado es que la aplicación queda pausada a la espera de que terminemos de atender el evento producido.

- **RSEG-01, no accederá a información sensible del usuario**: Al instalar la aplicación en nuestro dispositivo esta no nos solicita permisos de acceso para acceder a la información del usuario por lo que el sistema no puede acceder a

dicha información. Podemos ver la solicitud de permisos de nuestra aplicación en el manual de instalación descrito en el **apartado 4.2.2** de esta memoria.

- **RSEG-02, controlar la introducción de datos inconsistentes en la pantalla de configuración por parte del usuario**: El algoritmo de control de las barras de búsqueda de progreso descrito en el **apartado 3.2.4.4** se encarga de controlar los datos introducidos por el usuario.

Capítulo 6

Conclusiones y líneas futuras

A lo largo del presente capítulo analizaremos aquellas conclusiones que podemos extraer a posteriori del desarrollo de la aplicación, ventajas e inconvenientes de la misma. Además, plantearemos algunas mejoras o ampliaciones aplicables sobre nuestro sistema de forma que este pueda ser más preciso o incluya nuevas funcionalidades que proporcionen al usuario una experiencia más completa.

6.1 Conclusiones

Durante la realización del presente proyecto nos hemos ido familiarizando con las ventajas que supone la conducción eficiente. Hemos visto que aplicando unas instrucciones básicas podemos ahorrar mucho combustible, esto implica, de manera directa, una disminución sustancial del gasto en este aspecto además de las ventajas asociadas sobre el entorno natural. Empleando menos combustible conseguimos ahorrar dinero y disminuir la contaminación ambiental y acústica.

Debido a todo esto, es muy importante difundir las técnicas de la conducción eficiente entre la población, si la mayoría de los conductores practicaran la conducción eficiente se disminuiría de manera muy significativa el gasto de combustibles y, por tanto, la dependencia del petróleo, además, en términos de contaminación, significaría una reducción en la emisión de gases contaminantes que influiría positivamente en el entorno natural y en la propia salud de los conductores que lo apliquen junto con los demás ciudadanos.

Con el objeto de ayudar a difundir estas técnicas de la conducción eficiente y concienciar a los conductores en la importancia de llevarlas a cabo se ha creado nuestra aplicación. El sistema desarrollado informa al usuario en tiempo real acerca de las técnicas de la conducción eficiente, según el IDAE. En concreto, la aplicación ayuda al usuario en las siguientes tareas:

- Llevar engranada la marcha de la caja de cambios correcta en cada situación.
- Evitar aceleraciones demasiado pronunciadas.
- No hacer un uso excesivo del pedal del freno, aprovechando más el freno motor.
- Mantener una velocidad uniforme durante el trayecto.
- Mantener una velocidad moderada.
- Informar de la variación del consumo en función de la pendiente de la vía.
- Informar al usuario de que es recomendable apagar el motor en detenciones mayores de 60 segundos.

Además de la información en tiempo real, la aplicación pone a disposición del usuario aquella información más relevante sobre la conducción eficiente, de esta manera, el usuario puede acceder a ella fácilmente y así tenerla presente durante el viaje.

Por otro lado, la plataforma desarrollada permite que el usuario aprenda a corregir aquellas manías o tendencias que afectan negativamente al consumo de su vehículo, para este fin se ha desarrollado la pantalla de resumen del viaje.

A la vista de los resultados obtenidos (mostrados en el **capítulo 5** de la presente memoria) podemos concluir que la aplicación realiza un trabajo muy eficaz. En la prueba de comparación del consumo (**apartado 5.3**) se demostró que emplear correctamente nuestra aplicación implica un ahorro de combustible muy significativo, en la prueba realizada el ahorro resultante fue del 27.5%.

A modo de pequeña apreciación personal, la contundencia del resultado de dicha prueba a supuesto una evolución significativa en mi forma de conducir, primando ahora la conducción eficiente y la seguridad por encima del resto de factores.

Se puede otorgar mayor valor a los resultados obtenidos teniendo en cuenta la reducida lista de datos de los que la aplicación dispone y la poca precisión de los mismos, a priori, ofrecida por el dispositivo GPS. Precisamente, corregir la falta de precisión de los datos ofrecida por el sistema GPS ha supuesto un gran esfuerzo durante el desarrollo y un notable incremento en la complejidad del código fuente de la aplicación.

A pesar de la complejidad del código fuente debido a la falta de precisión de los datos, el procesamiento de los datos realizado por la plataforma se puede considerar una ventaja, ya que este es totalmente transparente al usuario y permite al sistema mostrar un informe de la situación actual del vehículo muy cercano a la realidad, incluso en algunos casos, como vimos en el **capítulo 5**, mostrando datos con más precisión que los propios indicadores del vehículo. Un buen diagnóstico de la situación real del vehículo en cada instante es clave, ya que este es el punto de partida de la funcionalidad de nuestra aplicación.

La ventaja más importante que ofrece la aplicación consiste en que esta permite al usuario aprender y aplicar de manera muy sencilla las técnicas de la conducción eficiente mientras conduce su propio vehículo.

El hecho de que la plataforma ayude al usuario a aplicar el concepto de la "conducción eficiente" conlleva consigo todas las ventajas asociadas a realizar este tipo de conducción, como son:

- Aumento en la seguridad.
- Ahorro de combustible.
- Ahorro del gasto debido al combustible.
- Menores costes de mantenimiento del vehículo.

Hoy en día, la falta de conciencia sobre la conducción eficiente en países como España supone que circular por una vía a una velocidad inferior al resto de vehículos o emplear aceleraciones suaves en una vía urbana puede suponer un obstáculo para el resto del tráfico rodado. Para ello, podemos emplear la adaptabilidad de la aplicación

para ser menos rigurosos en las técnicas de la conducción eficiente, a la espera de que el resto del tráfico rodado evolucione paulatinamente en esa dirección.

Otra ventaja muy interesante de nuestra aplicación es que se ha desarrollado de manera que sea compatible con todo tipo de tamaños de pantallas y densidades de estas. Este aspecto es muy importante debido a la gran heterogeneidad de los dispositivos Android disponibles en el mercado. Para lograr este objetivo se han tenido en cuenta los consejos del manual de desarrollador de Android en el apartado "mejores prácticas" (del inglés "best practices"). **[14]**

Por otro lado, con objeto de compensar el alto consumo de energía asociado al uso del dispositivo GPS, se ha dotado a nuestra aplicación de un interfaz que prima la funcionalidad por encima de la apariencia. Así, hemos conseguido que el consumo de batería de nuestra aplicación no sea desorbitado, sobre todo si lo comparamos con otras aplicaciones Android, muy populares, destinadas al entretenimiento o al intercambio de mensajes que, pese a su sencillez, poseen unos interfaces gráficos muy complejos que consumen la batería del dispositivo a mayor velocidad que nuestra aplicación.

Por último, el uso de nuestra aplicación durante la conducción puede contribuir a la distracción del conductor sobre su tarea principal, conducir, es conocido que la mayor parte de los accidentes se producen por pérdidas de atención. El conductor puede tener la tentación de observar la pantalla de su Smartphone para comprobar sus progresos durante un período de tiempo excesivamente largo, suponiendo esta actitud un aumento del riesgo de sufrir accidentes. Es por eso, que el uso de avisos sonoros de la aplicación es tan importante y, por consiguiente, se ha prestado especial atención a ese aspecto durante el desarrollo.

6.2 Líneas futuras

Durante este apartado vamos a describir algunas posibles mejoras para la aplicación que puedan ser implementadas en el futuro.

6.2.1 Nuevos idiomas

Ya que las claves de la conducción eficiente son comunes a todos los países, una de las posibles mejoras es la adición de algunos idiomas al interfaz, como pueden ser el inglés, alemán, etc.

Android posee herramientas que permiten identificar el idioma que tiene el dispositivo configurado por defecto y, a partir de este, mostrar el idioma de la aplicación adecuado. En ese caso, habría que definir un archivo de etiquetas de la aplicación

strings.xml para cada idioma. Por otro lado, sería necesario también grabar los mensajes de voz en los mismos idiomas.

Con previsión de introducir esta mejora en el futuro, todas las etiquetas y textos incluidos en la aplicación se encuentran en el archivo citado, de esta forma, bastaría con traducir dicho archivo al idioma deseado para cambiar el idioma de los interfaces de nuestra aplicación.

6.2.2 Acceder a información real del vehículo

En aquellos vehículos que dispongan de una centralita de diagnóstico de a bordo (OBD2), mediante un adaptador Bluetooth, conectado a dicha centralita, podríamos conectar nuestra aplicación a ella y obtener los datos reales del vehículo, como son las revoluciones del motor, la marcha engranada, la velocidad del velocímetro, etc.

A partir de estos datos, podríamos ser más precisos en las recomendaciones sobre el cambio de marchas del vehículo. Aplicando la información de las RPM del motor sobre el cambio de marchas en las deceleraciones e incluso introduciendo las recomendaciones del IDAE sobre los cambios de marchas en pendientes.

Mediante este dispositivo Bluetooth podríamos prescindir del GPS para calcular la mayoría de los datos, aunque seguiríamos necesitándolo para calcular la pendiente, además, perderíamos precisión en el cálculo de la velocidad si no empleásemos los datos del GPS, debido al error de precisión del velocímetro.

Podríamos hacer que coexistieran ambas formas de obtener los datos y que fuera el usuario el que decidiera cual de las dos prefiere utilizar. De esta manera, la aplicación seguiría siendo compatible con todos los Smartphones que posean GPS, como lo es ahora, y además, incluiríamos esta opción para usuarios que quieran profundizar más en las claves de la conducción eficiente.

6.2.3 Añadir nuevas opciones de configuración

Se pueden introducir nuevas opciones de configuración en la pantalla inicial, como pueden ser:

- Numero de marchas del vehículo.
- Peso del vehículo.
- Configuraciones por defecto en función del trayecto (para trayectos urbanos, por autovías o una combinación de ambos).

En el primer caso, la aplicación adaptaría la relación de marchas preestablecida al caso de que el vehículo posea una caja de cambios de 5 velocidades o de 6 velocidades,

ya que la mayoría de los vehículos nuevos de alta cilindrada, sobre todo los que poseen motores diesel, poseen caja de cambios con 6 marchas.

Por otro lado, en los cálculos de la variación del consumo debido a la pendiente, se ha empleado un peso estándar, de este modo, se considera igual la variación del consumo independientemente del vehículo, con un peso adaptable la aplicación podría ser un poco más precisa en esos cálculos.

El último caso permitiría al usuario introducir el tipo de vía que va a predominar durante el viaje y, de esta forma, la aplicación aportaría la configuración recomendada en cada caso. En un trayecto urbano, los límites de velocidad serían más bajos que en un trayecto por autovía. Del mismo modo, en un trayecto urbano la aplicación podría ser un poco más permisiva con los límites de aceleración, mientras que circulando por una autovía la velocidad debería ser lo más uniforme posible y, así, los límites de aceleración serían más restrictivos.

6.2.4 Mejora del interfaz de usuario

Una buena mejora consistiría en diseñar un interfaz de usuario más atractivo para este. En la versión actual de la aplicación a primado la eficacia sobre la belleza, sería muy interesante conseguir aunar eficacia y belleza para que la aplicación creciera en popularidad entre los usuarios de dispositivos Android. Además, sería especialmente importante que este nuevo interfaz no aumentara sensiblemente el consumo de batería del dispositivo.

6.2.5 Reducir el tiempo que tarda la aplicación en iniciarse

La aplicación puede tardar unos minutos en comenzar a recibir datos, debido al sistema GPS. Sería muy interesante reducir el tiempo de espera inicial hasta que la aplicación empieza a mostrar los datos del viaje de cara a no hacer esperar al usuario inútilmente.

6.2.6 Mejora del algoritmo de procesamiento de datos de altitud

Como se ha comentado en diversos apartados de este proyecto, son necesarios 5 datos de altitud para generar un solo valor de altitud con mejor precisión. Se ha elegido este algoritmo por la sencillez y, por tanto, la rapidez con la que se lleva a cabo en tiempo de ejecución. Es posible mejorar este algoritmo haciéndolo un poco más complejo.

Por ejemplo, una posible mejora de este algoritmo consistiría en calcular la primera altitud con las 5 primeras muestras de altitud recibidas, la segunda altitud la podríamos calcular empleando las 4 últimas muestras de altitudes empleadas para el primer cálculo y el nuevo valor recibido, la tercera altitud emplearía las cuatro últimas muestras del segundo caso y la siguiente recibida, así sucesivamente. De esta manera calcularíamos una altitud por cada nueva coordenada recibida.

Capítulo 7

Planificación y presupuesto

En este capítulo estudiaremos la planificación del presente proyecto y elaboraremos un presupuesto para llevar a cabo el mismo.

7.1 Planificación del proyecto

Dentro de este apartado incluiremos, en primer lugar, la planificación inicial del proyecto y, después, compararemos esta con la planificación final obtenida después de la realización del proyecto.

Para estudiar la planificación del proyecto se ha realizado un diagrama de Gantt, este diagrama está dividido en seis tareas principales, que a su vez, se componen de tareas más sencillas. Estas son las tareas de las que se compone el proyecto, ordenadas de forma temporal, es decir, la primera tarea es la primera a realizar:

- **Formación inicial (documentación previa)**: En esta fase, el autor del proyecto ha adquirido los conocimientos necesarios para poder programar una aplicación compleja en Android. Además, dentro de esta tarea se incluye la instalación de las herramientas necesarias para el desarrollo de aplicaciones y la resolución de algunos ejemplos consistentes en desarrollar aplicaciones sencillas. Durante esta fase de documentación previa está incluida la labor de investigación sobre conducción eficiente.

- **Análisis de requisitos**: Se trata de decidir qué queremos que haga la aplicación (sus funcionalidades) y como queremos que lo haga. Sobre estas decisiones tomadas inicialmente se llevará a cabo toda la implementación de la plataforma.

- **Diseño de la aplicación**: En esta fase se diseña toda la estructura interna del sistema, además de los interfaces de usuario y todos los algoritmos necesarios para realizar el trabajo descrito en la tarea anterior.

- **Implementación y pruebas unitarias de la aplicación**: En esta tarea se transforman en código fuente los diseños de la tarea anterior, estos se desarrollarán por separado de forma que se puedan realizar pruebas independientes (pruebas unitarias) para cada elemento desarrollado antes de integrar el conjunto.

- **Pruebas de sistema**: Durante esta fase se realizan pruebas del sistema completo con el objetivo de verificar que este cumple con todos los objetivos descritos en el apartado de requisitos.

- **Documentación**: Esta tarea consiste en realizar toda la documentación del presente proyecto, desde comentar con precisión el código fuente hasta realizar la presente memoria.

Antes de mostrar el diagrama realizado (**tablas 7.1**, **7.2**, **7.3** y **7.4**), hay que tener en cuenta que todas las tareas están realizadas por el mismo individuo (el autor del proyecto), además, la dedicación diaria a dicho proyecto es de entre 3 y 4 horas.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Formación inicial (documentación previa)**	**30 días**	**lun 03/09/12**	**lun 15/10/12**
2	Formación en Android	20 días	lun 03/09/12	vie 28/09/12
3	Instalación y pruebas herramientas desarrollo	5 días	lun 01/10/12	vie 05/10/12
4	Investigación conducción eficiente	5 días	lun 08/10/12	lun 15/10/12
5	**Análisis de requisitos**	**5 días**	**mar 16/10/12**	**lun 22/10/12**
6	Casos de uso	2 días	mar 16/10/12	mié 17/10/12
7	Requisitos del sistema	3 días	jue 18/10/12	lun 22/10/12
8	**Diseño de la aplicación**	**12 días**	**mar 23/10/12**	**jue 08/11/12**
9	Arquitectura del sistema	2 días	mar 23/10/12	mié 24/10/12
10	Interfaces de usuario	2 días	jue 25/10/12	vie 26/10/12
11	Algoritmo entrada datos de usuario	3 días	lun 29/10/12	mié 31/10/12
12	Algoritmo recepción y procesado datos GPS	2 días	vie 02/11/12	lun 05/11/12
13	Algoritmo procesamiento datos de altitud	1 día	mar 06/11/12	mar 06/11/12
14	Algoritmo cálculo variación consumo debido a la pendiente	2 días	mié 07/11/12	jue 08/11/12
15	**Implementación y pruebas unitarias de la aplicación**	**35 días**	**mié 14/11/12**	**lun 07/01/13**
16	Arquitectura general del sistema	10 días	mié 14/11/12	mar 27/11/12
17	Interfaces de usuario	6 días	mié 28/11/12	mié 05/12/12
18	Algoritmo entrada datos de usuario	3 días	jue 06/12/12	lun 10/12/12
19	Algoritmo recepción y procesado datos GPS	8 días	mar 11/12/12	jue 20/12/12
20	Algoritmo procesamiento datos de altitud	2 días	mié 02/01/13	jue 03/01/13
21	Algoritmo cálculo variación consumo debido a la pendiente	2 días	vie 04/01/13	lun 07/01/13
22	**Pruebas de sistema**	**9 días**	**mié 09/01/13**	**lun 21/01/13**
23	Pruebas en el emulador	2 días	mié 09/01/13	jue 10/01/13
24	Pruebas de consumo en circuito real	4 días	vie 11/01/13	mié 16/01/13
25	Pruebas de funcionalidad y robustez	3 días	jue 17/01/13	lun 21/01/13
26	Documentación	60 días	mar 22/01/13	mar 16/04/13

Tabla 7.1. Diagrama de Gantt inicial 1/4

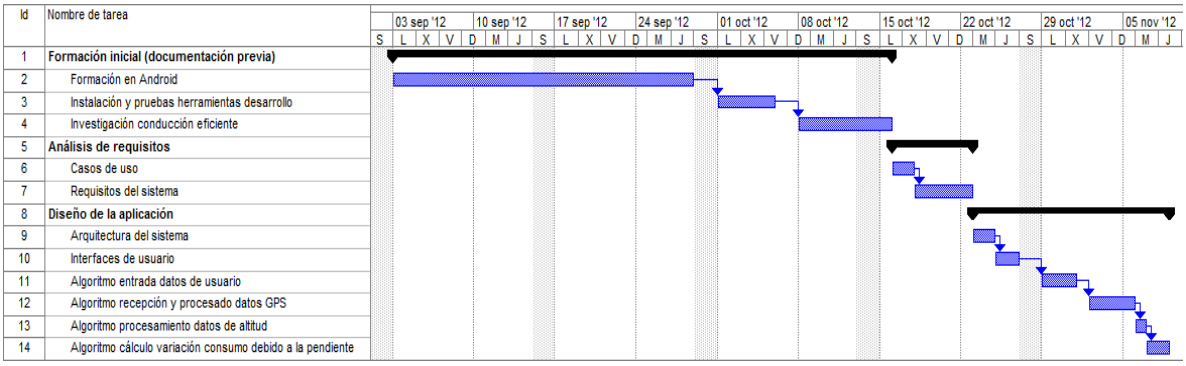
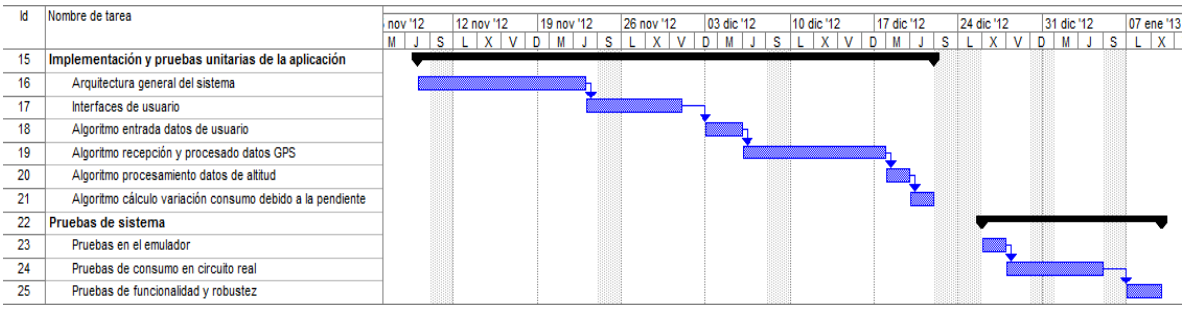
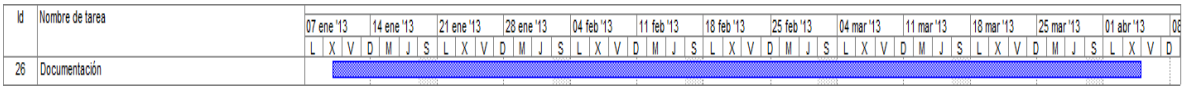


Tabla 7.2. Diagrama de Gantt inicial 2/4



Id	Nombre de tarea
15	**Implementación y pruebas unitarias de la aplicación**
16	Arquitectura general del sistema
17	Interfaces de usuario
18	Algoritmo entrada datos de usuario
19	Algoritmo recepción y procesado datos GPS
20	Algoritmo procesamiento datos de altitud
21	Algoritmo cálculo variación consumo debido a la pendiente
22	**Pruebas de sistema**
23	Pruebas en el emulador
24	Pruebas de consumo en circuito real
25	Pruebas de funcionalidad y robustez

Tabla 7.3. Diagrama de Gantt inicial 3/4



Id	Nombre de tarea
26	Documentación

Tabla 7.4. Diagrama de Gantt inicial 4/4

Como se puede observar en las **tablas 7.1**, **7.2**, **7.3** y **7.4**, esta es la planificación inicial detallada del presente proyecto. Todas las tareas se encuentran ordenadas de forma secuencial ya que están realizadas por una sola persona. Una buena forma de reducir el tiempo de realización del proyecto consistiría en realizar algunas tareas en paralelo, menos la tarea de documentación previa y la tarea de análisis de requisitos todas las demás tareas tienen partes que se pueden realizar en paralelo, empleando para ello varias personas en el mismo equipo de trabajo.

Vamos ahora a mostrar el diagrama de Gantt resultante al término del presente proyecto, de esta forma podremos observar las diferencias entre la planificación inicial y la final, se justificará cada una de las diferencias surgidas en la planificación. Primero mostraremos la tabla de tareas (**tabla 7.5**) y posteriormente el diagrama de Gantt (**tablas 7.6**, **7.7** y **7.8**):

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Formación inicial (documentación previa)**	**30 días**	**lun 03/09/12**	**lun 15/10/12**
2	Formación en Android	20 días	lun 03/09/12	vie 28/09/12
3	Instalación y pruebas herramientas desarrollo	5 días	lun 01/10/12	vie 05/10/12
4	Investigación conducción eficiente	5 días	lun 08/10/12	lun 15/10/12
5	**Análisis de requisitos**	**5 días**	**mar 16/10/12**	**lun 22/10/12**
6	Casos de uso	2 días	mar 16/10/12	mié 17/10/12
7	Requisitos del sistema	3 días	jue 18/10/12	lun 22/10/12
8	**Diseño de la aplicación**	**17 días**	**mar 23/10/12**	**jue 15/11/12**
9	Arquitectura del sistema	2 días	mar 23/10/12	mié 24/10/12
10	Interfaces de usuario	2 días	jue 25/10/12	vie 26/10/12
11	Algoritmo entrada datos de usuario	3 días	lun 29/10/12	mié 31/10/12
12	Algoritmo recepción y procesado datos GPS	2 días	vie 02/11/12	lun 05/11/12
13	Algoritmo procesamiento datos de altitud	1 día	mar 06/11/12	mar 06/11/12
14	Algoritmo cálculo variación consumo debido a la pendiente	7 días	mié 07/11/12	jue 15/11/12
15	**Implementación y pruebas unitarias de la aplicación**	**43 días**	**vie 16/11/12**	**lun 21/01/13**
16	Arquitectura general del sistema	12 días	vie 16/11/12	lun 03/12/12
17	Interfaces de usuario	10 días	mar 04/12/12	lun 17/12/12
18	Algoritmo entrada datos de usuario	3 días	mar 18/12/12	jue 20/12/12
19	Algoritmo recepción y procesado datos GPS	10 días	vie 21/12/12	mié 09/01/13
20	Algoritmo procesamiento datos de altitud	3 días	jue 10/01/13	lun 14/01/13
21	Algoritmo cálculo variación consumo debido a la pendiente	5 días	mar 15/01/13	lun 21/01/13
22	**Pruebas de sistema**	**9 días**	**mar 22/01/13**	**vie 01/02/13**
23	Pruebas en el emulador	2 días	mar 22/01/13	mié 23/01/13
24	Pruebas de consumo en circuito real	4 días	jue 24/01/13	mar 29/01/13
25	Pruebas de funcionalidad y robustez	3 días	mié 30/01/13	vie 01/02/13
26	Documentación	70 días	lun 04/02/13	mié 15/05/13

Tabla 7.5. Diagrama de Gantt final 1/4

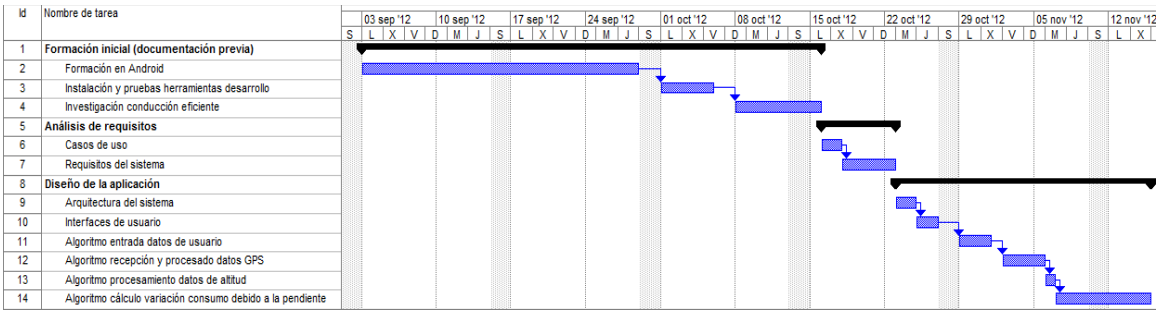


Tabla 7.6. Diagrama de Gantt final 2/4

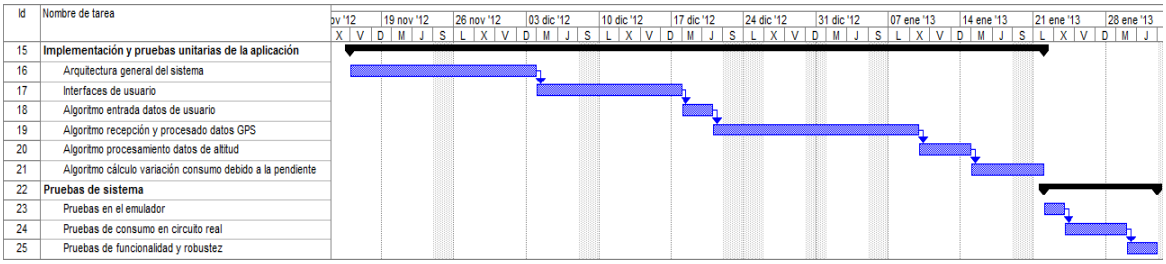


Tabla 7.7. Diagrama de Gantt final 3/4

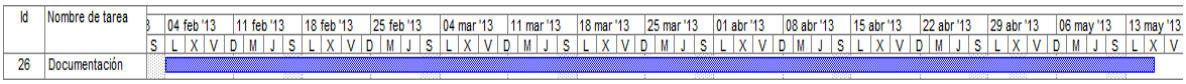


Tabla 7.8. Diagrama de Gantt final 4/4

A continuación, nos vamos a centrar en aquellas tareas que han sufrido algún tipo de cambio en la planificación, describiendo para cada una de ellas los motivos de dicho cambio:

- **Algoritmo cálculo variación consumo debido a la pendiente (Id = 14)**: Esta tarea ha sufrido un retraso de 5 días respecto a la previsión inicial. Muy posiblemente estamos hablando de la tarea más compleja de todo el proyecto. En la previsión inicial se contemplaba la posibilidad de aplicar una investigación existente publicada sobre este tema (ajena a nuestro proyecto). No ha sido posible encontrar una investigación que aporte una relación directa entre consumo y pendiente, como consecuencia, se ha tenido que realizar dicha investigación antes de desarrollar el correspondiente algoritmo, además, algunas estimaciones iniciales mostraron errores durante la fase de pruebas teóricas y tuvieron que ser desestimadas.

- **Arquitectura general del sistema (Id = 16)**: Esta tarea ha sufrido un retraso de 2 días respecto a la previsión inicial. El retraso sufrido en esta tarea es debido, principalmente, a la falta de experiencia del desarrollador en aplicaciones Android, los mayores problemas surgidos durante esta tarea han estado relacionados con el contenido del archivo manifiesto.

- **Interfaces de usuario (Id = 17)**: Esta tarea ha sufrido un retraso de 4 días respecto a la previsión inicial. En esta ocasión han surgido problemas asociados al aspecto de los interfaces en función de los distintos tamaños y densidades de pantalla. Por otro lado, también se ha tenido que resolver la variación del tamaño de pantalla disponible en función de si el menú se muestra normalmente o cómo una barra de acción (variación del aspecto en función del API instalado en el dispositivo).

- **Algoritmo recepción y procesado datos GPS (Id = 19)**: Esta tarea ha sufrido un retraso de 2 días respecto a la previsión inicial. Estos dos días de retraso se debieron a la falta de precisión de los datos obtenidos respecto a la distancia entre coordenadas. Fue necesario hacer una comprobación imprevista entre algunos métodos proporcionados por el API de Android para averiguar cuál de ellos ofrece una precisión mejor. Además, fue necesario corregir las imprecisiones presentes, menores de 1 metro, cuando el dispositivo se encuentra inmóvil.

- **Algoritmo procesamiento datos de altitud (Id = 20)**: Esta tarea ha sufrido un retraso de 1 día respecto a la previsión inicial. El retraso en esta tarea es debido al emulador de Android, este no permite introducir altitudes en las coordenadas

generadas, como consecuencia, fue necesario generar un algoritmo que se encargara de establecer las altitudes adecuadas en cada coordenada.

- **Algoritmo cálculo variación consumo debido a la pendiente (Id = 21)**: Esta tarea ha sufrido un retraso de 3 días respecto a la previsión inicial. El retraso en esta tarea está asociado al retraso de la tarea con Id = 14. Hubo una versión anterior del algoritmo que estima la variación del consumo que superó las pruebas teóricas pero no consiguió superar las pruebas unitarias, de este modo, hubo que realizar los cambios necesarios y volver a realizar las pruebas unitarias.
- **Documentación (Id = 26)**: Esta tarea ha sufrido un retraso de 10 días respecto a la previsión inicial. En gran medida, este retraso es debido a la variación del tamaño de la memoria respecto a la estimación inicial. La previsión fue que serían necesarias 150 páginas para describir en detalle el proyecto realizado, finalmente se han superado con creces ese número de páginas.

7.2 Presupuesto

En este apartado calcularemos la inversión necesaria para realizar el proyecto. Para calcular el presupuesto final tendremos en cuenta la planificación inicial, no la final, puesto que el presupuesto se realizó anteriormente al desarrollo de la aplicación y a la realización de la presente memoria. Los costes asociados a la diferencia entre la planificación inicial y la planificación resultante serán asumidos por el desarrollador y no por el cliente.

Para calcular el coste del personal sabemos que, según la planificación inicial, en total, se dedicarán 151 días. Además, sabemos que sólo se encargará una persona de la realización del proyecto y que esta dedicará 4 horas diarias.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DEL PROYECTO

1.- Autor

Javier Rodríguez Guijarro

2.- Departamento

Departamento de Ingeniería Telemática

3.- Descripción del proyecto

- Título:
  Diseño e implementación de una aplicación Android que permita ahorrar combustible en vehículos.
- Duración (meses): **8**
- Tasa de costes indirectos: **20%**

4.- Presupuesto total del proyecto

El presupuesto total del proyecto es de **15.154,32 €.**

5.- Desglose presupuestario (costes directos)

PERSONAL						
Apellidos y nombre	**NIF**	**Categoría profesional**	**Dedicación (hombres mes) a)**	**Coste hombre mes (€)**	**Coste (€)**	**Firma de conformidad**
Javier Rodríguez Guijarro		Ingeniero	4,62	2.694,39	12.448	
				Total	**12.448**	

a) 1 hombre mes = 131,25 horas

Tabla 7.9. Presupuesto costes de personal

EQUIPOS					
Descripción	**Coste (€)**	**% Uso dedicado proyecto**	**Dedicación (meses)**	**Periodo de depreciación (meses)**	**Coste imputable (€) b)**
Ordenador portátil Dell Inspiron Dual Core	599	75	8	48	74,87
Smartphone Samsung Galaxy Ace	150	50	8	36	16,66
				Total	**91,53**

b) Fórmula del cálculo de la amortización:

(A / B) · C · D

A = los meses que el equipo es utilizado para este cometido
B = periodo de depreciación del equipo (48 meses el ordenador portátil y 36 meses el Smartphone)
C = coste del equipo (sin IVA)
D = % de tiempo que se emplea el equipo durante el periodo de utilización en este proyecto

Tabla 7.10. Presupuesto costes de equipos

OTROS COSTES DIRECTOS DEL PROYECTO		
Descripción	**Empresa**	**Coste imputable (€)**
Alquiler Volkswagen Passat 2.0 i durante 6 horas	Europcar	66
Cuota de registro como desarrollador	Google	19,26
	Total	**85,26**

Tabla 7.11. Presupuesto otros costes directos

6.- Resumen de costes

Concepto	Presupuesto costes totales (€)
Personal	12.448
Amortización	91,53
Otros costes directos	85,26
Costes indirectos (20%)	2.529,53
Total	**15.154,32**

Tabla 7.12. Resumen de costes

En la **tabla 7.12** se muestra el coste total del proyecto, esta cantidad ha sido calculada utilizando los subtotales mostrados en las **tablas 7.9**, **7.10** y **7.11**. Por lo tanto, el presupuesto total de este proyecto asciende a la cantidad de **QUINCE MIL CIENTO CINCUENTA Y CUATRO CON TREINTA Y DOS EUROS**.

Leganés, a de de 2013

El ingeniero proyectista

Fdo. Javier Rodríguez Guijarro

7.3 Comercialización de la aplicación

Gracias a "Google Play" tenemos dos formas de recuperar la inversión inicial de nuestro proyecto. Por un lado, tenemos la opción de establecer un precio de venta para la aplicación, en ese caso, "Google Play" tiene fijada una tarifa de transacción del **30%**, es decir, de cada venta que se haga de la aplicación "Google Play" se queda el 30%.

Por ejemplo, podríamos recuperar la inversión vendiendo **15.155** copias de nuestra aplicación a **1€** cada una, debido a la tarifa de transacción necesitamos un 30% más de ventas para obtener la cantidad necesaria, en nuestro caso, necesitaríamos vender **19.702** copias de nuestra aplicación para recuperar la inversión inicial. A partir de ese número de descargas comenzaríamos a obtener beneficios.

También podemos optar por que nuestra aplicación se descargue de manera gratuita e incluir publicidad en ella. Para poder obtener ingresos provenientes de la publicidad necesitamos que la aplicación se haga popular, cuantas más descargas se produzcan de nuestra aplicación más dinero obtendremos de la publicidad. Es por esta razón que la tendencia en "Google Play" es invertir en marketing de la plataforma que ofertamos y de esta manera hacer que más usuarios conozcan su existencia.

Para que podamos incluir publicidad en nuestra aplicación, Google nos proporciona una librería para Android llamada **Admob**. Esta librería es muy sencilla de utilizar, sólo necesitamos declarar un objeto **Adview** en nuestros archivos de vistas y añadir en el método ***onCreate*()**, de la actividad a la que pertenece dicha vista, una llamada al método encargado de solicitar la publicidad al servidor de Google.

Glosario de acrónimos

GPS	*Global Positioning System*
SO	*Sistema Operativo*
RIM	*Research In Motion*
IOS	*Iphone OS*
INE	*Instituto Nacional Estadística*
CO_2	*Dióxido de Carbono*
CO	*Monóxido de Carbono*
UE	*Unión Europea*
g	*gramo y aceleración de la gravedad*
Kg	*Kilogramo*
AVEN	*Agencia Valenciana de ENergia*
IDAE	*Instituto para la Diversificación y Ahorro de la Energía*
rpm	*Revoluciones Por Minuto*
PC	*Personal Computer*
ADT	*Android Development Kit*
SDK	*Software Development Kit*
JDK	*Java Development Kit*
OpenGL	*Open Graphics Library*
GSM	*Global System for Mobile*
EDGE	*Enhaced Data Rates for GSM Evolution*
3G	*Tercera Generación*
API	*Application Programming Interface*
SSL	*Secure Sockets Layer*

ID	*Identificador*
VM	*Virtual Machine*
SMS	*Short Message Service*
SD	*Secure Digital*
URI	*Uniform Resource Identifier*
XML	*eXtensible Markup Language*
MIME	*Multipurpose Internet Mail Extensions*
HTML	*HyperText Markup Language*
CPU	*Central Processing Unit*
JVM	*Java Virtual Machine*
OBD2	*On Board Diagnostics*
MPEG	*Moving Picture Experts Group*
MP3	*MPEG audio layer III*
m	*masa*
N	*Newtons*
J	*Julios*
h	*altura*
E_{mec}	*Energía mecánica*
E_c	*Energía cinética*
E_p	*Energía potencial*
USB	*Universal Serial Bus*
ppp	*Píxeles Por Pulgada*

Referencias

[1] Our mobile planet: Global Smartphone Users. Disponible [Internet]: <http://services.google.com/fh/files/blogs/final_global_smartphone_user_study_2012.pdf>. Accedido en mayo de 2013.

[2] Kantar world panel emtech. Disponible [Internet]: <http://www.kantarworldpanel.com/kwp_ftp/global/comtech/Kantar_Worldpanel_ComTech_Smartphone_OS_barometer_11_7_12.pdf>. Accedido en mayo de 2013.

[3] Los hogares españoles recortan un 1% de gastos en 2011 salvo en alquiler y suministros. Disponible [Internet]: <http://www.telemadrid.es/?q=node/156978/>. Accedido en mayo de 2013.

[4] El consumo de la gasolina y gasóleo se hunde un 12,3%. Disponible [Internet]: <www.elmundo.es/elmundo/2013/04/02/economia/1364903047.html>. Accedido en mayo de 2013.

[5] Consumo de carburante y las emisiones de CO2. Disponible [Internet]: <www.idae.es/Coches/portal/Consumo.aspx>. Accedido en mayo de 2013.

[6] Contaminación del aire: el problema son los coches, no el polvo sahariano. Disponible [Internet]: <www.ecologistasenaccion.org/article10289.html>. Accedido en mayo de 2013.

[7] Efecto invernadero. Disponible [Internet]: <www.profesorenlinea.cl/Ciencias/Efecto_invernadero.htm>. Accedido en mayo de 2013.

[8] Guías de calidad del aire – actualización mundial de 2005. Disponible [Internet]: <www.who.int/phe/health_topics/outdoorair_aqg/es/index.html>. Accedido en mayo de 2013.

[9] Contaminación ecológica causada por el uso masivo de automóviles. Disponible [Internet]: <www.elpais.com/diario/1980/08/16/sociedad/335224807_850215.html>. Accedido en mayo de 2013.

[10] Manual de conducción eficiente para vehículos turismo (AVEN 2007). Disponible [Internet]: <http://web.ua.es/es/ecocampus/documentos/campanas/movilidad/manual-conduccion-eficiente.pdf>. Accedido en mayo de 2013.

[11] Definición de Android. Disponible [Internet]: <www.conceptodefinicion.de/android>. Accedido en mayo de 2013.

[12] Platform Versions. Disponible [Internet]: <developer.android.com/about/dashboards/index.html>. Accedido en mayo de 2013.

[13] Arquitectura de Android. Disponible [Internet]: <androideity.com/2011/07/04/arquitectura-de-android/>. Accedido en mayo de 2013.

[14] App Fundamentals. Disponible [Internet]: <developer.android.com/guide/components/fundamentals.html>. Accedido en mayo de 2013.

[15] Características del lenguaje Java. Disponible [Internet]: <www.iec.csic.es/criptonomicon/java/quesjava.html>. Accedido en mayo de 2013.

[16] Volere, plantilla de especificación de requisitos, edición 11, febrero 2006. Disponible [Internet]: <www.volere.co.uk/pdf files/template_es.pdf>. Accedido en mayo de 2013.

[17] Android <activity>. Disponible [Internet]: <developer.android.com/guide/topics/manifest/activity-element.html>. Accedido en mayo de 2013.

[18] Building and Running from Eclipse with ADT. Disponible [Internet]: <developer.android.com/tools/building/building-eclipse.html>. Accedido en mayo de 2013.

[19] La UMA constata que los velocímetros de los coches marcan más velocidad de la real. Disponible [Internet]: <www.diariosur.es/v/20111026/malaga/constata-velocimetros-coches-marcan-20111026.html>. Accedido en mayo de 2013.